NPS ARCHIVE
2000.09
ABABNEH, M.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NETWORK CONFIGURATION USING XML

by

Mohammad Ababneh

September 2000

Thesis Advisor: Geoffrey Xie
Associate Advisor: Daniel Dolk

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: **Network Configuration Using Xml**			5. FUNDING NUMBERS
6. AUTHOR: Mohammad Ababneh			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The primary goal of this thesis is to investigate the use of the eXtensible Markup Language (XML) as a network configuration language. Network configuration is a difficult and time-consuming task. Current network configuration solutions are based on proprietary configuration languages and parsers. XML is a platform-neutral data representation language and worldwide standard. It is potentially advantageous to use XML to configure networks, however, XML was not developed for network configuration. A new XML based configuration solution for the Server and Agent Active Network Management System (SAAM) is provided to marshal evidence that XML can be used effectively as a network configuration language.

14. SUBJECT TERMS XML, Quality of Service, Network Configuration, Next Generation Internet,Networks			15. NUMBER OF PAGES 138
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

NETWORK CONFIGURATION USING XML

Mohammad Ababneh
1ST Lieutenant, Royal Jordanian Air Force
B.S., Mu'tah University, 1994

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT
and
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

NPS ARCHIVE
2000.09
ABABNEH, M.
Thesis
A1136
C.1

THIS PAGE INTENTIONALLY LEFT BLANK

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

The primary goal of this thesis is to investigate the use of the eXtensible Markup Language (XML) as a network configuration language. Network configuration is a difficult and time-consuming task. Current network configuration solutions are based on proprietary configuration languages and parsers. XML is a platform-neutral data representation language and worldwide standard. It is potentially advantageous to use XML to configure networks, however, XML was not developed for network configuration. A new XML based configuration solution for the Server and Agent Active Network Management System (SAAM) is provided to marshal evidence that XML can be used effectively as a network configuration language.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
1. QoS Internet 1
2. The XML Revolution 2
B. THESIS SCOPE 3
C. METHODOLOGY 4
D. MAJOR CONTRIBUTION OF THIS THESIS 5
E. ORGANIZATION 5

II. THE SERVER AND AGENT ACTIVE NETWORK MANAGEMENT SYSTEM (SAAM) 7
A. WHAT IS SAAM? 7
1. Best-Effort Service 7
2. Integrated Service 8
3. Differentiated Service 8
B. SAAM ARCHITECTURE 8
1. The SAAM Server 9
2. The SAAM Router 10
3. The SAAM DemoStation 10

III. THE XML SOLUTION DESIGN 13
A. WHAT IS XML? 13
1. XSL 14
2. XML 15
3. DTD 16
4. XML IDE's 16
5. XML Parsers 17
a. DOM 18
b. SAX 18
6. XML Applications 18
B. WHY USE XML IN SAMM? 19
1. Flexibility 20
2. Reusability 20
3. World Standard 21
4. Interoperability 21
5. Robustness 21
6. Portability 22
C. THE XML SOLUTION DESIGN AND IMPLEMENTATION 22
1. The XML IDE 22
a. Well-formedness 23
b. Validation 23
2. The XMLSPY IDE Features 24

a. Textual and Tabular Views of the XML Files 24
b. Automatic DTD File Building 25
D. THE DEVELOPMENT OF THE SAAM CONFIGURATION XML FILE 26
E. THE DEVELOPMENT OF THE SAAM CONFIGURATION DTD FILE 29
1. The DTD Rules 29
2. Generating the DTD File 29
F. SUMMARY 31
IV. THE DEVELOPMENT OF THE JAVA PARSER 33
A. HOW TO ACCESS THE XML DOCUMENT THROUGH JAVA 33
1. DOM 33
2. SAX 34
3. Why SAX? 34
B. THE APACHE XERCES PARSER 34
C. THE DEMOSTATION JAVA PROGRAM 35
1. SAXParserDemo Class: public class SAXParserDemo {} 36
2. MyContentHandler Class: class MyContentHandler implements ContentHandler {} 38
3. MyErrorHandler: class MyErrorHandler implements ErrorHandler 44
D. INTEGRATION AND TESTING 45
1. Integration 45
2. Testing 46
a. Ease of use 46
b. User Interface 47
c. Portability 47
d. Performance 47
E. SUMMARY 48
V. THE CONCLUSION AND FUTURE WORK 49
A. CONCLUSIONS 49
B. LESSONS LEARNED 49
1. SAX vs. DOM 49
2. XML IDE 50
3. XML in Networking 50
C. FUTURE WORK 50
1. The Implementation of the PIB 51
2. The Use of XML to Exchange Messages (Packets) 51
3. The Development of a Web-Based Network Management System 51
4. The Development of Smart Self-Configurable Nodes 52
APPENDIX A. XML SAX PARSER DEMO STATION SOURCE CODE 53
APPENDIX B. THE APACHE XML SAX PARSER SOURCE CODE 67

APPENDIX C. THE APACHE XML SAX PARSER – CONTENT HANDLER SOURCE CODE 93
APPENDIX D. THE APACHE XML SAX PARSER – ERROR HANDLER SOURCE CODE 101
APPENDIX E. THE SAAM CONFIGURATION DTD FILE 105
APPENDIX F. THE XML CONFIGURATION FILE 107
APPENDIX G. A USER GUIDE TO USE XMLSPY IN SAAM CONFIGURATION 109
LIST OF REFERENCES 115
INITIAL DISTRIBUTION LIST 117

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 2.1. The Hierarchical Design of SAAM. [Ref. 2] 9
Figure 3.1. The Relationship between XML, HTML and SGML. 14
Figure 3.2. An Example XML Document. 15
Figure 3.3. An Example DTD File. 16
Figure 3.4. The XMLSPY 3.0.7 IDE. 17
Figure 3.5. The Tabular View of the XML File – the First Part. 24
Figure 3.6. The Tabular View of the XML File – the Second Part. 25
Figure 3.7. The One-Click DTD Generation. 26
Figure 3.8. The SAAM XML Network Configuration File. 29
Figure 3.9. The DTD File Developed for the SAAM Configuration. 30
Figure 4.1. A UML Diagram of the DemoStation Program. 36
Figure 4.2. The FileChooser Object. 45
Figure 4.3. The Final Results from the Parsing Program. 48

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 3.1 Recurrence Operators. 31

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge Prof. Geoffery Xie for his guidance and enthusiasm without which I couldn't complete this research. I would like also to acknowledge the support and guidance of Prof. Daniel Dolk throughout this thesis.

I am gratefull to my small family. I appreciate the support of my wife Huda and daughter Leen during my research and writing of this thesis. I am also gratefull for my parents Ahmed and Fatima Ababneh for their love and sacrifice during all the past years.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The purpose of this thesis is to provide evidence that the eXtensible Markup Language (XML) can be employed as a useful network configuration language. The applications of XML do not end simply with web development, data interchange and databases. Rather, XML can make network configuration and management much easier than currently possible.

1. QoS Internet

The Internet started as the ARPANET in 1969 with only data in mind. As the Internet evolved, new requirements for this network were introduced. Among those new requirements were voice and video capabilities over the Internet. The original Internet was not designed to support these applications. The Internet provides a best-effort service, which means there is no guarantee that a packet sent over the Internet will reach its destination. Further, there is no guarantee which path it is going to take to reach its final destination, or when it will get to its destination. This limitation of the Internet makes it difficult for applications that need better service such as voice, video and other types of critical data.

These limitations have led people to look for solutions and new designs that can support Quality-of-Service (QoS). The Internet routing and protocol system should be redesigned to support QoS. Some people call this the "Next Generation Internet" initiative (NGI) or "Internet version 2". This new version of the Internet should be able to support QoS, or in other words, the new Internet should be able to transfer data, voice,

and video with high quality and provide guarantees of high quality service for its customers.

Several QoS models [Ref. 1] have been developed recently to address this issue. The first one is Integrated Service (IntServ), which is characterized by resource reservation. The second one is Differentiated Service (DiffServ). The third is Multi-Protocol Label Switching (MPLS). The last two are characterized by relative QoS.
A more detailed discussion about these alternatives will follow in the next chapter.

The original design of the Internet gives full control to network routers. There is no central command that watches and guides the routers. This results in a significant traffic flow, especially for QoS systems. In order to send a stream of voice or video over the Internet in a consistent and timely manner, central control and QoS metrics adopted by network routers should exist. *The Server and Agent Active Network Management system* (SAAM) adopts this design. The main idea of SAAM is to use off-line servers that have control and routers that comply with servers. Chapter II contains more details about SAAM.

2. The XML Revolution

The end of the 20th century witnessed the birth of the open source and universal standardization of the Information Technology field . The first IT universal revolution was TCP/IP (Transmission Control Protocol / Internet Protocol). This protocol made it possible to connect the whole world in one Internetwork (Internet) leading to universal networking. The second revolution was the development of Java, which is a universal programming language. The third breakthrough was HTML (Hyper Text Markup

Language), which enabled universal browsing and introduced the thin client concept to the IT dictionary.

The fourth revolution, which inspired this thesis, is XML (eXtensible Markup Language). It is a markup language like the others but it has had an extraordinary effect on IT. The main contribution of XML is universal data formatting. The use of XML as the low-level data format makes it easy for any application in the world to communicate with others, regardless of boundaries and limitations. XML removes the operating system, the character set and the application barriers, which are the traditional reasons behind non-interoperability. Also, its support of UNICODE (Universal Code) gives it an advantage that will enable this data format to be made universal as quickly as possible.

The question we address is why is XML important to QoS networks? Our hypothesis is that XML will make it easier to store configuration data and service metrics on network routers and servers, and make them available to any network management system that needs this information. This is very useful in a new network management system design like SAAM.

B. THESIS SCOPE

The primary goal of this thesis is to investigate the use of XML as a network configuration language. Network configuration is a difficult and time-consuming task. Current network configuration solutions are based on proprietary configuration languages and parsers. XML is a platform-neutral data representation language and worldwide standard. It is potentially advantageous to use XML to configure networks, however, XML was not developed for network configuration. A new XML based configuration

solution for SAAM is provided to marshal evidence that XML can be used effectively as a network configuration language.

C. METHODOLOGY

I started the development of the XML configuration solution by studying the previous configuration approaches. The first approach was to embed the configuration information into a Java program. This was an inflexible approach for doing the configuration. A new Java program was needed for each new topology. The second approach was using ASCII files to store configuration information and a Java program to parse the information. The ASCII file contains special syntax and keywords that make it difficult to create new topologies and there was no tool to validate and verify the configuration. Any syntax error might exit the parser or produce an error, which results in incomplete configuration process.

I have started the development of the XML solution for the Server and Agent Active Network Management System (SAAM) by developing the XML files and the DTD files. A thorough study was conducted to find out all the possible configuration elements that can be used and what are the elements used by each type of the SAAM components. This phase was very important, because based on this the structure of the DTD file was developed. The DTD file enforces the structure of the XML file and works as a reference to validate it.

After the development of the XML and DTD files was done and verified, I started the development of the Java parsing program. This program will take the XML file as its input and extract the configuration information from it. It will configure the network by sending all the configuration pieces to the nodes that it belongs to.

D. MAJOR CONTRIBUTION OF THIS THESIS

This thesis provides a proof-of-concept of a network management system that utilizes XML as the configuration data and metrics storage medium on the network devices. It also contributes to SAAM by making the network configuration and topology establishment easy, portable and supportive of the latest XML technology. This will make SAAM portable to many environments and platforms.

E. ORGANIZATION

This thesis is divided into the following chapters chapters.

1. Chapter II discusses the SAAM environment.
2. Chapter III reviews the basics of XML, why to use XML in this thesis and presents the design of the XML files to be used in the SAAM system.
3. Chapter IV presents the Java implementation of the XML solution.
4. Chapter V contains the conclusions and suggests future work to be done with XML in SAAM.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THE SERVER AND AGENT ACTIVE NETWORK MANAGEMENT SYSTEM (SAAM)

A. WHAT IS SAAM?

The goal of SAAM according to Varble and Yarger is to "find a solution that will provide a guaranteed QoS while still maintaining the simplicity and robustness of the underlying TCP/IP architecture." [Ref. 2]. SAAM seeks to provide this QoS through multiple levels of services, central network management and area division into service regions. SAAM has two major components: the SAAM Server and the SAAM Router. The server has control over all routing and quality services in its area while the router is just a point where service can be requested and executed [Ref. 3].

The design of SAAM had taken into account three types of services in order to provide QoS:

1. Best-Effort Service

This type of service is the traditional Internet service. The Internet and its protocols had been developed to provide Best-Effort service. There are no guarantees for a client requesting service to get the sufficient bandwidth for transmission, nor are there any guarantees that the packets of a client will arrive at the destination at all or they will arrive within a specific time frame. Time critical applications on the Internet currently suffer from this situation.

2. Integrated Service

The Integrated Service, unlike the Best-Effort Service, provides guaranteed QoS. The user receives the quality of service requested for an application or session. The individual user flow specifies the minimum bandwidth required and the maximum delay and packet loss that it needs. Integrated Service is characterized by resource reservation. The service provider has to set up paths and reserve resources before the user packets can be sent.

3. Differentiated Service

The Differentiated Service, unlike the Integrated Service, offers the user a session at a predetermined level of service by negotiating with that user. The service provider commits itself to the initial agreement with a customer, but does not provide any guarantee regarding additional loads or services. There are different classifications of packets requesting Differentiated Service and there are different queues for each class of service at each output link, unlike the Integrated Service, which allocates one queue for each session.

B. SAAM ARCHITECTURE

SAAM is a hierarchical structure of servers and routers. The SAAM network is divided into regions that have a regional server and up to 40 routers controlled by the server. Those regional servers report to higher level servers that have control over all the regional servers. This central management gives the SAAM system the advantage of having control over the entire network and enables the applications to have one point of reference to obtain QoS service. Instead of negotiating with local routers or regional servers for service that they cannot guarantee, the service request is sent to the top server where it can make the correct decision about providing the service. Also, this design

reduces the processing power requirements on the router side. Figure 2.1 illustrates the hierarchical structure of SAAM.

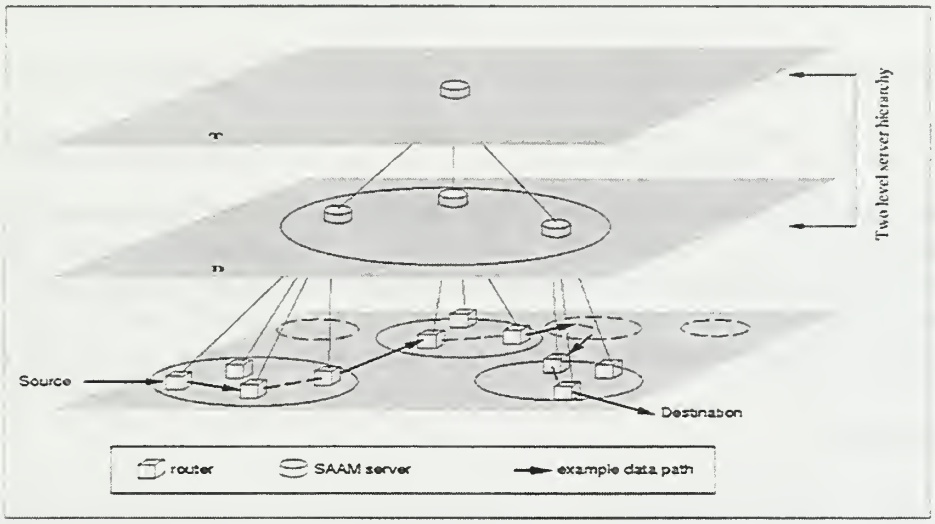


Figure 2.1. The Hierarchical Design of SAAM. [Ref. 2]

There are two major components in SAAM:

1. The SAAM Server

The SAAM server is the real network manager. There is a Primary SAAM Server in each region. This server is backed up by a Backup Server to provide fault tolerance. The main responsibility of the server is to assist its regional routers with routing and other network management tasks. All router information such as the addresses, interfaces, bandwidth and performance are kept and maintained by the server in a special database called "PIB" (Path Information Base). Once all the information is available, the server is capable of making decisions on service requests by clients. The server will be able to assign a path to match the end-to-end QoS metrics requested by a client. Also, the server

is able to change paths (re-route) if the QoS required is not achievable for one reason or another, or there is a better QoS match on another path.

2. The SAAM Router

The SAAM Routers are the reception points for service requests by network clients. All request applications enter the network through the routers, which forwards the requests to the server for approval. The server will respond to each request either by assigning a path to meet the request or by denying the request. When a path is assigned to a client for the first time, the server will also send route-update messages to all routers in the path to install appropriate entries to their routing tables to create the packet-forwarding path.

Each router is responsible for reporting its status (i.e., the state of its interfaces) to the server so that the server always has an accurate view of the network to make good routing decisions. Due to the hierarchical structure of the SAAM system, that status information first goes up to the regional server, then aggregated and forwarded by the regional server to servers higher in the hierarchy. This reduces the traffic on the network for the purpose of the router status update, and keeps all decision makers informed of the status of each router.

3. The SAAM DemoStation

The DemoStation is the component that initializes the SAAM test topology. It sends all the configuration information to the servers and routers in the topology. In the first design, each topology was hard-coded in a Java DemoStation program. This requires a new Java program to be developed every time in order to test a new topology. To address this shortcoming, an ASCII file-based configuration approach was developed. In the ASCII file-based configuration, different topologies are stored in different ASCII files

and they are parsed by a standard Java program.. This approach provided more flexibility than the hard-coded approach. However, it has some limitations. It is hard to learn the special configuration language that was developed for this purpose. Also, there were no means for syntax and structure validation of the ASCII files. Furthermore, the interoperability could not be achieved between different platforms and operating systems using this kind of configuration.

The latest approach for the Demo Station configuration was developed in XML and serves as the core of this thesis. A more detailed discussion of why XML was used and its benefits are discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. THE XML SOLUTION DESIGN

A. WHAT IS XML?

XML stands for eXtensible Markup Language, a W3C (World Wide Web Consortium) standard markup language that was adopted in December 1999. It is a much simpler language than its predecessor SGML (Standard Generalized Markup Language) and has fewer base tags than SGML. It is also different from HTML (Hyper Text Markup Language), which is another W3C standard that was also derived from SGML. The main difference between XML and HTML is that in HTML there are a static, predefined number of tags that can be used to develop an HTML document. Any other tags are simply ignored by the browsers trying to parse these documents. There are only a few predefined tags in XML.

The major power of XML is that the user has the flexibility to define his own tags to describe data. A parser capable of reading an XML file can access the meta-data and the data represented by these user-defined tags. In HTML, on the other hand, the parser (browser) only translates these tags into a good representation in the browser's GUI without paying attention to the real value of the data.

The figure below shows the relationship between the markup languages.

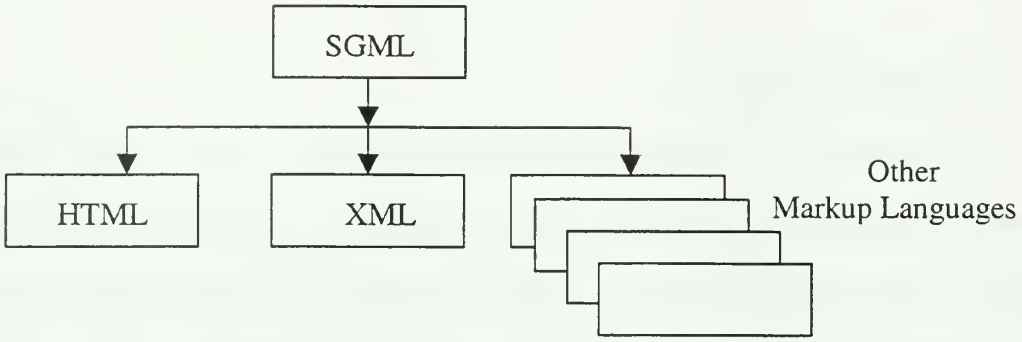


Figure 3.1. The Relationship between XML, HTML and SGML.

When we are talking about XML that doesn't mean only the XML document, but also a set of other basic concepts in the XML specification. Some of these are:

1. XSL

XML itself is about data while HTML is about representation. XSL (extensible Style Sheet) is the W3C standard for XML content representation that can be used to view XML documents. For any single XML document, there can be many XSL schemas or representations for displaying that document. One XSL can be used to display the document in a browser with some criteria such as displaying the data sorted by name, number or income. Another XSL can display the data in a different order according to another criterion and a third one can display the XML document on a cellular phone screen or on a PDA (Personal Digital Assistant). For our research, we don't need to use XSL. It would be important if we wanted to display the XML content through a browser, but we will develop a Java parser instead to read the document and obtain the configuration data.

2. XML

The XML document itself consists of the XML code, which includes the tags or the structure of the document, and the data itself. The XML document structure can be thought of as a tree of nodes or as a large object that contains sub-objects each of which contains elements. Other than the XML headers, the user is free to define his own tags but within a structure that is enforced by the DTD (Document Type Definition), or schema for this XML file. An example of an XML document is included below. Writing an XML document will be covered in more detail when we discuss the XML-based SAAM network configuration in Chapter V.

```
<? xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0 NT (http://www.xmlspy.com) by mababneh (nps) -->
<!DOCTYPE SAAM_Net SYSTEM "C:\XML\SAAM_1.dtd">
<SAAM_Net>
  <Node>
      <NodeType>PrimaryServer</NodeType>
      <NodeName>Server A</NodeName>
      <IPv4>131.120.8.137</IPv4>
      <ServerFlowID>1</ServerFlowID>
      <TimeScale>350</TimeScale>
      <SC_MetricType>0</SC_MetricType>
      <SC_CycleTime>200</SC_CycleTime>
      <SC_GlobalWaitTime>250</SC_GlobalWaitTime>
      <SC_LocalWaitTime/>
  </Node>
  <Node>
      <NodeType>Router</NodeType>
      <NodeName>Router A</NodeName>
      <IPv4>131.120.8.147</IPv4>
      <TimeScale>350</TimeScale>
      <InterfaceAddressType>IPv6</InterfaceAddressType>
      <Agent>PreviousNodeProbe</Agent>
  </Node>
  <Node>
      <NodeType>Router</NodeType>
      <NodeName>Router B</NodeName>
      <IPv4>131.120.9.66</IPv4>
      <TimeScale>350</TimeScale>
      <InterfaceAddressType>IPv6</InterfaceAddressType>
      <Agent>PreviousNodeProbe</Agent>
      <Agent>NextNodeProbe</Agent>
  </Node>
</SAAM_Net>
```

Figure 3.2. An Example XML Document.

3. DTD

The Document Type Definition is a separate structural file that enforces the rules to which an XML document must conform. The DTD file itself contains the user defined elements and their attributes. In this file, the elements are arranged in such a way to represent how the structure of the XML file should appear. This structure is a usually tree-like with one root element that contains other elements. A core concept in XML design is the validation of the XML document against the DTD file. If the XML file is valid, then it is useable; otherwise it is not. This validation concept is different from the well-formedness concept, which checks the syntax of the XML file within itself. The following listing shows an example of a DTD file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!–edited with XML Spy v3.0 NT (http://www.xmlspy.com) by mababneh (nps) →
<!–DTD generated by XML Spy v3.0 NT (http://www.xmlspy.com)-->
<!ELEMENT SAAM_Net (Node+)>
<!ELEMENT Node (NodeType, NodeName, Ipv4, ServerFlowID?, TimeScale, SC_MetricType?,
SC_CycleTime?, SC_GlobalWaitTime?, SC_LocalWaitTime?, Agent*)>
<!ELEMENT Ipv4 (#PCDATA)>
<!ELEMENT NodeName (#PCDATA)>
<!ELEMENT NodeType (#PCDATA)>
<!ELEMENT ServerFlowID (#PCDATA)>
<!ELEMENT TimeScale (#PCDATA)>
<!ELEMENT SC_MetricType (#PCDATA)>
<!ELEMENT SC_CycleTime (#PCDATA)>
<!ELEMENT SC_GlobalWaitTime (#PCDATA)>
<!ELEMENT SC_LocalWaitTime EMPTY>
<!ELEMENT InterfaceAddressType (#PCDATA)>
<!ELEMENT Agent (#PCDATA)>
```

Figure 3.3. An Example DTD File.

4. XML IDE's

Finding a mature Integrated Development Environment tool for XML development was not easy due to the fact that XML is still evloving. New standards and improvements to XML and DTD are introduced frequently. While writing this thesis, for example, there were a few IDE tools that claimed to support a comprehensive IDE for XML. Most people develop their XML documents through ASCII editors. This approach

is difficult and requires attention to spelling and the structure of the document. Some of the newly developed IDEs provide a GUI that enables the user to see the XML document as a table or a set of graphical nodes (objects) with their attributes annotated. There is no perfect IDE yet for XML because the standards have not yet been decided upon. This situation is similar to that of HTML in its infancy. There was no IDE that could support HTML perfectly and most of the HTML developers had to write their HTML documents using ASCII editors. The following figure shows a GUI of the XML-IDE used in this thesis, which is XML SPY 3.0.7 developed by Icon Information Systems. Figure 3.4 shows the XML SPY version of the XML file example in Figure 3.2 as a table.

5. XML Parsers

In an application development effort using a high level programming language, the parser is the most basic and important component to the developer. It sits between the application (e.g., Java program) and the XML document. It simplifies the process of parsing the XML document in trying to get useful information. Instead of developing a

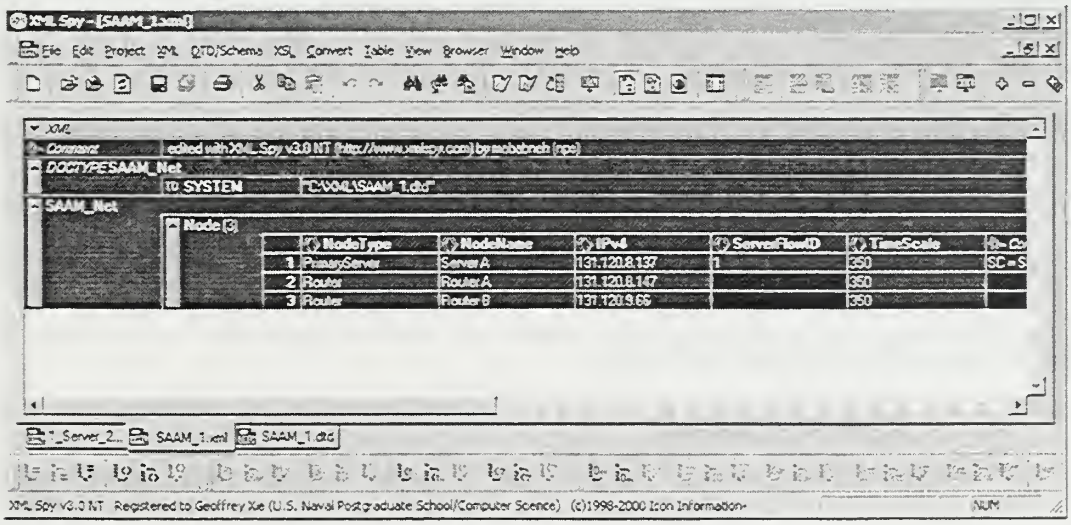


Figure 3.4. The XMLSPY 3.0.7 IDE.

non-reusable parsing program for each XML document, the parser can do the parsing job through a standard interface that is easy and reusable. There are two major types of XML parsers: DOM (Document Object Model) and SAX (Simple API for XML).

a. DOM

The DOM parser reads the XML document and converts it to a tree of nodes (objects). It provides methods that can go up and down the tree hierarchy once the XML document is converted to a tree of objects. This makes it very easy to use in XML document parsing. For example, a node can be checked for the elements that it has and a simple method like: *getNodeName* or *getNodeValue* can be used to retrieve the data. The disadvantage of DOM is that the entire document must reside in the system's memory. This, in the case with a large number of nodes such as a very large network topology, can cause insufficient memory problems. A further discussion will be provided in Chapter VI.

b. SAX

SAX is an event driven approach to XML document parsing. It goes through the document sequentially, executing or calling certain methods when certain elements of the XML document are reached during the parsing process. For example, while reading the XML file sequentially, when a new element is encountered, then a certain event can occur such as getting the value of that element. This approach does not require much memory and is the developer's choice when parsing an XML document with a large number of nodes in the tree.

6. XML Applications

Everybody is beginning to think of XML as a new web technology that is going to replace HTML as the web authoring language. That is only part of the truth. In fact, XML can be used for other applications. One of these applications is a unified, flexible,

and powerful configuration language. When configuration data is put into XML documents on any kind of computing system, this provides a very sound approach to configuring the system. Instead of having the configuration data hard-coded in the configuration program, or stored in a platform-dependent ASCII, EBCDIC or UNICODE file that needs special parsing programs, the system can find the configuration information in standard, platform-independent, structured and readily-available parsers.

B. WHY USE XML IN SAMM?

The current SAAM DemoStation uses ASCII files to store topology information. Before this approach, each topology had to be built into a separate DemoStation Java program. This approach lacks flexibility and reusability of the code.

Ababneh, Brunstad and Gaines developed the SAAM ASCII configuration approach [Ref. 4]. It makes the configuration process more flexible and easier to use than the hard-coding approach. Each topology is built in a separate ASCII file, allowing the DemoStation Java program to remain stable and reusable. Whenever a new topology is needed, then a new ASCII file is created to capture this topology. This was an improvement over the previous approach, but it still has some limitations. The contents of the ASCII file should consist of certain key words, white spaces and comments. All of these components should be written correctly following a syntax especially designed for this purpose. The developers called this syntax SCL+ (SAAM Configuration Language plus). The problem with this approach is that there is no quick way to ensure that the ASCII file has been written correctly. So, if one line violates the syntax, then the DemoStation program will fail to parse the topology file. Another problem is that the

DemoStation must run on a platform that supports ASCII; this limitation might not be desirable when SAAM reaches its distribution phase.

This lack of uniformity and standardization has led to a search for a new technique for SAAM network configuration. XML was seen as the best replacement. Our experience suggests that using XML for network configurationwill be standard practice in the next few years. Some of the features of XML and its relationship with the SAAM configuration are described below.

1. Flexibility

XML is a very flexible language. It supports user-defined tags while enforcing the structure and correctness of the XML document. The rules that exist in the DTD file guarantee a syntactically and semantically error free document. It is similar to a compiler approach.

Two main features of XML are relevant: well-formedness and validity [Ref. 5]. Well-formedness refers to the correctness of the XML syntax such as tag naming, beginning, terminating, comments, etc. The validity property refers to the concept of validating the XML document against the DTD file to ensure that the XML document structure follows user-defined rules and restrictions. DTD specifies whether an element is mandatory, how many elements can exist in one node and what attributes these elements possess.

2. Reusability

For an application that needs data to change each time, it is absolutely necessary to find a vehicle such as XML, which separates the configuration data from the working code. It then becomes possible to obtain a reusable program that can interact with as

many XML files as required. The problem of hard-coding the topology in the configuration program is that it prohibits code reuse. XML definitely solves this problem, since one reusable Java program can be written to read any XML file.

3. World Standard

XML is a W3C standard. It was developed under the specification number REC-xml-19980210 [Ref. 6]. Obviously, using it as a configuration language will present no interoperability problems. Unlike other configuration approaches which require learning a new set of instructions and then being restricted by them, XML can be accepted anywhere in the world without locking the user into a proprietary approach.

4. Interoperability

XML is indeed interoperable. It is a worldwide standard that can be used anywhere regardless of the platform. An XML document will be the same on any operating system and hardware platform. This gives the SAAM configuration the capability to run on any platform. Whether the XML file is stored locally on the configuration machine, or stored on an URI (Unified Resource Identifier), it can be easily accessed and processed. For example, the Java Demostation program can run on a Windows NT box while the XML file resides on a Linux box. The default use of the UNI-CODE in XML contributes to the interoperability feature.

5. Robustness

The contents of the XML file can always be validated against a DTD file. This is an important feature of the XML design and is called Validity. This feature allows for the discovery of any error in the syntax or the structure of the nodes in the topology. In the ASCII approach, there is no such capability. If there is a syntax error in one of the key words being looked for, then the entire topology might not be established and much time

would be wasted debugging the text file. In XML, there is always a reference that can be checked for correctness and robustness.

6. Portability

XML is as portable as Java. A Java program can be run on any system that has the Java Virtual Machine (JVM) installed on it. An XML file is the same, readable on any platform. This makes a perfect match: Java as the portable code and XML as the portable data.

C. THE XML SOLUTION DESIGN AND IMPLEMENTATION

Several steps are required to develop the proposed XML solution for a SAAM configuration. The development of the XML solution consists of two major steps: development of the XML file and development of the DTD file.

1. The XML IDE

At the time this thesis was being written, the W3C XML specifications were only several months old. Consequently no sophisticated IDE for XML exists, but some vendors were shipping promising products. After conducting an intensive search for an acceptable XML IDE, XMLSPY version 3.0 was selected as the best choice. Unlike the other tools available, it has a user-friendly interface that can display the XML file in multiple views, especially the table view with the ability to drill down a cell to represent sub-elements. Also, it has a strong feature of automatically generating DTD files after learning the pattern of element repetition in the XML file. The alternative to an IDE was to work with a text editor. The use of a text editor would be difficult because the XML syntax must always have the following two basic features.

a. Well-formedness

This feature signifies that the XML syntax should follow a certain order and use headers, comments, tags and data. All elements should be nested inside larger elements until they reach the root node. This nesting can go to any level of detail, which results in a tree of nodes and elements. The place where a reference to a DTD file exists should be clearly identified and the comments should be written correctly. The tags should start and end in such a way that all elements are between their correct tags [Ref. 7].

b. Validation

The W3C validation requirement is necessary to ensure that a certain XML file follows the rules and structure located in the associated DTD file. The DTD file enforces and restricts what can be written in the XML file. It is possible to add any number of elements and data in the XML file, but these element definitions should also be in the DTD file as should the number and type of each element that can be in an XML file. In addition, in the DTD file, optional elements can be specified, as can elements that are mandatory and elements of which there should be at least one. Without the DTD file, the XML file might be invalid, or in other words, an XML parser might not be able to read the XML file. A basic difference between XML and HTML is that HTML parsers (Browsers) can ignore invalid HTML syntax whereas XML parsers do not ignore syntax errors.

The validation process can be thought of as something similar to compilation. After writing the source XML syntax, the validation process serves as the compiler that indicates which lines have syntax or semantic errors in them.

2. The XMLSPY IDE Features

As mentioned previously, there was no mature XML IDE on the market, but XMLSPY provides some features that are very helpful.

a. Textual and Tabular Views of the XML Files

This feature allows the developer a choice of using a textual approach for developing the XML file or a very easy and powerful tabular approach. In the tabular approach, each node is represented by a row and each element is a cell in that row. It can also embed other tables representing elements that have sub-elements. At any time you can switch between textual and tabular views very easily. Figures 3.5 and 3.6 are the tabular view of the same XML file displayed in Figure 3.2

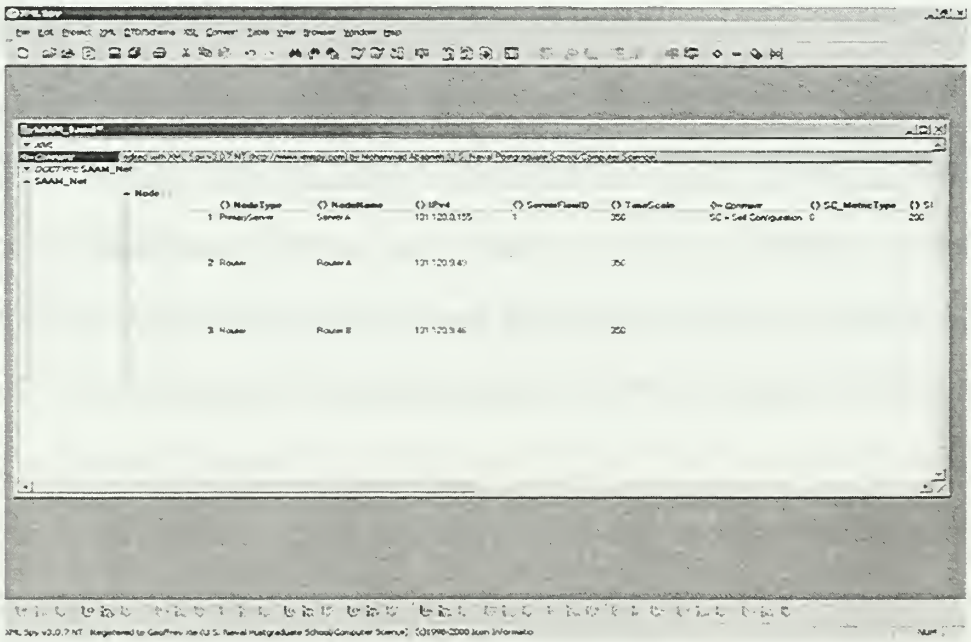


Figure 3.5. The Tabular View of the XML File – the First Part.

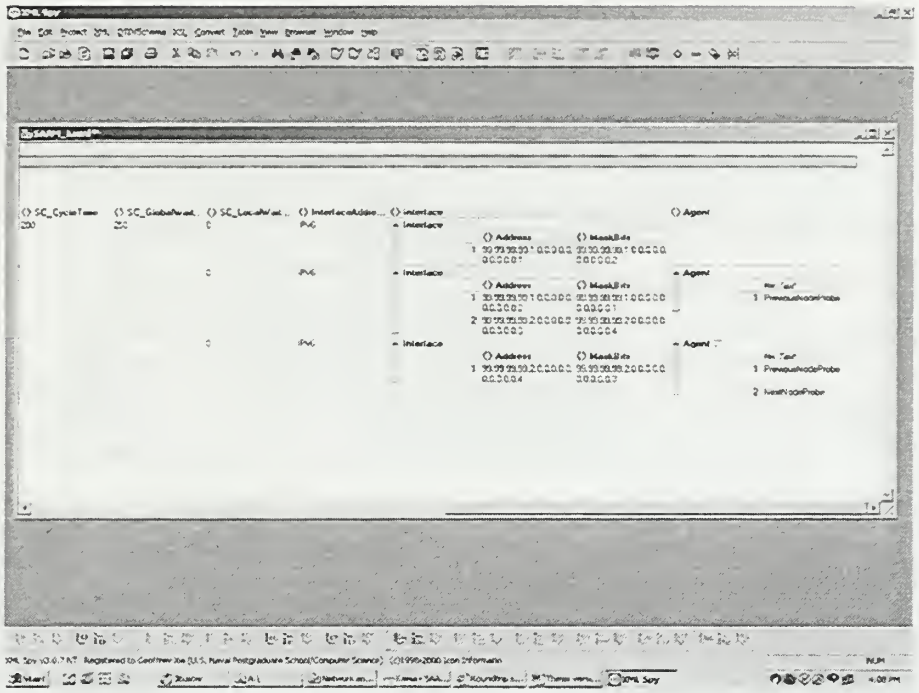


Figure 3.6. The Tabular View of the XML File – the Second Part.

b. Automatic DTD File Building

It is possible to start development by designing the DTD file, but it is necessary to return and modify the file frequently to accommodate all the cases encountered during the XML file design. This process takes a lot of time and effort. XMLSPY provides the capability of automatically generating a DTD file based on all the cases seen in the XML file. This methodology has proven to be very powerful. A sample XML file can be written that has all the known cases and then the DTD file is generated that can accommodate this XML file and all future possibly larger and more complicated XML files. It can learn by itself which rules to include, how the sequence is structured, and which elements are mandatory or optional. Figure 3.7 illustrates how to generate a DTD file.

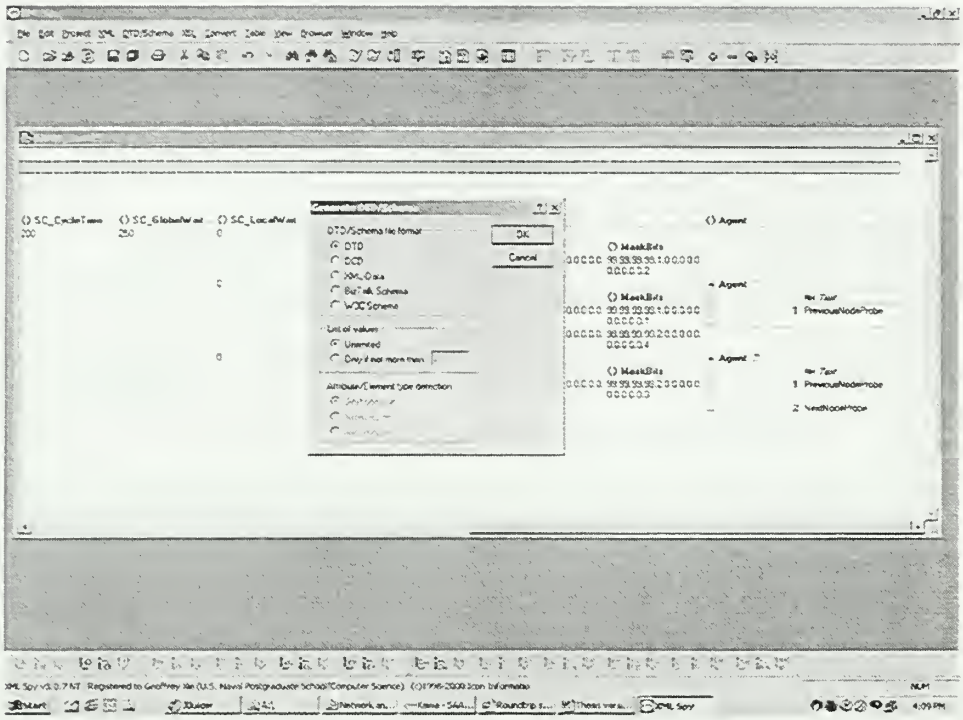


Figure 3.7. The One-Click DTD Generation.

D. THE DEVELOPMENT OF THE SAAM CONFIGURATION XML FILE

SAAM consists of three major types of nodes: Primary Server, Backup Server and Router. For each one, several variables must be known in order to stand up the node. Some of these variables are unique to a specific type of node and others are shared. A list of all the variables used to configure SAAM nodes follows.

- Node Type: This variable specifies the type of SAAM node. A Node can be a Primary Server, a Backup Server or a Router.
- Node Name: The name of this node in the topology
- IPv4: The IP address of machine hosting this SAAM node
- Server Flow ID: The flow ID used by this server node
- Time Scale: The time scale used by this node
- Self Configuration Metric Type: The metric type for self-configuration used by this server node [Ref. 8].
- Self Configuration Cycle Time: The self-configuration cycle time used by this server node

- Self Configuration Global Wait Time: The global wait time of a node for UCM messages from its children in a self-configuration cycle initiated by this server node
- Self Configuration Local Wait Time: The local wait time of a node for ParentNotification messages from its potential children in a self-configuration cycle initiated by this server node
- Interface Address Type: This variable was added to make the SAAM configuration more general. The Interface address type can be specified by this variable to represent any kind of protocol. It is now IPv6.
- Interface: This element represents an interface on this node and it consists of two sub-elements:
    - Address: The address of the interface –(The default is an IPv6 address)
    - MaskBits: The number of network bits in the subnet mask
- Agent: The class name of a resident agent that will be deployed to this node

As seen from the list above, some variables are used only by servers and others are used by both servers and routers.

The variables or elements (in the XML language) used in the XML file are as follows:

- NodeType
- NodeName
- IPv4
- ServerFlowID
- TimeScale
- SC_MetricType
- SC_CycleTime
- SC_GlobalWaitTime
- SC_LocalWaitTime
- InterfaceAddressType
- Interface: This element represents an interface on a node and it consists of two sub-elements:

- Address
- MaskBits

- Agent

The XML file representing a One Server Two Router topology is shown in Figure 3.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Mohammad Ababneh (U.S. Naval Postgraduate School/Computer Science) -->
<!DOCTYPE SAAM_Net SYSTEM "file:///C:/XML/SAAM_1.dtd">
<!-- The XML file contains the configuration information of the desired network topology. It makes sure that the structure is correct by
vakidating these contents with the DTD file, which is defined above

The SAAM configuration consists from elements which can contain other elements.The  main element is the node which represents
the servers and routers. each node has a set of elements and values.-->
<SAAM_Net>
  <Node>
    <NodeType>PrimaryServer</NodeType>
    <NodeName>Server A</NodeName>
    <IPv4>131.120.9.46</IPv4>
    <ServerFlowID>1</ServerFlowID>
    <TimeScale>350</TimeScale>
    <!--SC = Self Configuration-->
    <SC_MetricType>0</SC_MetricType>
    <SC_CycleTime>200</SC_CycleTime>
    <SC_GlobalWaitTime>250</SC_GlobalWaitTime>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
      <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1</Address>
      <MaskBits>40</MaskBits>
    </Interface>
  </Node>
  <Node>
    <NodeType>Router</NodeType>
    <NodeName>Router A</NodeName>
    <IPv4>131.120.8.155</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
      <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2</Address>
      <MaskBits>40</MaskBits>
    </Interface>
    <Interface>
      <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.3</Address>
      <MaskBits>40</MaskBits>
    </Interface>
    <Agent>PreviousNodeProbe</Agent>
  </Node>
  <Node>
    <NodeType>Router</NodeType>
    <NodeName>Router B</NodeName>
    <IPv4>131.120.9.49</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
```

```
        <InterfaceAddressType>IPv6</InterfaceAddressType>
        <Interface>
            <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.4</Address>
            <MaskBits>40</MaskBits>
        </Interface>
        <Agent>PreviousNodeProbe</Agent>
        <Agent>NextNodeProbe</Agent>
    </Node>
</SAAM_Net>
```

Figure 3.8. The SAAM XML Network Configuration File.

E. THE DEVELOPMENT OF THE SAAM CONFIGURATION DTD FILE

As already seen, there are several rules to which a SAAM configuration XML file should comply. The DTD file should establish these rules in order to make the XML configuration valid.

1. The DTD Rules

- Each node in SAAM requires mandatory elements. NodeType, NodeName, IPv4 and TimeScale are mandatory and there should be only one for each of these elements.
- Some of the elements can be included in the Server node, but not the Router node. SC_GlobalWaitTime, SC_LocalWaitTime, SC_CycleTime and SC_MetricType are exclusively for the servers. Only one value of each element should be included for a Server node. These values are required to instantiate a server, but they are not required by routers. Thus, a node can have zero or one of these elements.
- All nodes are required to have at least one interface element or it cannot participate in any topology. Thus, for each SAAM node, there should be one or more Interface elements.
- Each Interface element should have only one Address element and one MaskBits element.
- Each node can have zero or more Agent elements.

2. Generating the DTD File

The DTD file can be developed manually by following the syntax of the DTD files and by applying the rules discussed above. This process takes a lot of time and effort and provides no guarantees that this DTD file will validate each possible case of the XML content. Additionally, it takes time to learn the DTD syntax.

The IDE tool that I have used eliminates all these complications of DTD development. The DTD file can be generated automatically by reading through the XML file. With just one click of a button a DTD file will be generated with the guarantee that it will enforce the rules found in the XML file. One thing that should be taken into account is that the XML file has to be correct. If it is not, then the DTD is wrong, which in turns causes the XML files to be invalid. To use this feature, the XML developer has to be sure that he is generating a correct DTD from a correct XML. In this case, the XML file is used as an example by which the IDE learns the rules in generating a DTD file. After DTD file is generated and verified that it really represents the XML file, it can be used to validate unlimited number of XML files. So, the user should include in the example XML file all the possible arrangements of elements, names and values that might be in a node. Figure 3.4 shows the DTD code developed for the SAAM configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XML Spy v3.0.7 NT (http://www.xmlspy.com)-->
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Agent (#PCDATA)>
<!ELEMENT IPv4 (#PCDATA)>
<!ELEMENT Interface (Address, MaskBits)>
<!ELEMENT InterfaceAddressType (#PCDATA)>
<!ELEMENT MaskBits (#PCDATA)>
<!ELEMENT Node (NodeType, NodeName, IPv4, ServerFlowID?, TimeScale,
SC_MetricType?,
SC_CycleTime?, SC_GlobalWaitTime?, SC_LocalWaitTime, InterfaceAddressType,
Interface+, Agent*)>
<!ELEMENT NodeName (#PCDATA)>
<!ELEMENT NodeType (#PCDATA)>
<!ELEMENT SAAM_Net (Node+)>
<!ELEMENT SC_CycleTime (#PCDATA)>
<!ELEMENT SC_GlobalWaitTime (#PCDATA)>
<!ELEMENT SC_LocalWaitTime (#PCDATA)>
<!ELEMENT SC_MetricType (#PCDATA)>
<!ELEMENT ServerFlowID (#PCDATA)>
<!ELEMENT TimeScale (#PCDATA)>
```

Figure 3.9. The DTD File Developed for the SAAM Configuration.

As seen from the previous code, three lines describe the structure of the DTD file.

- <!ELEMENT SAAM_Net (Node+)>

This indicates that the root element of the XML file consists of several nodes.

- <!ELEMENT Node (NodeType, NodeName, IPv4, ServerFlowID?, TimeScale, SC_MetricType?, SC_CycleTime?, SC_GlobalWaitTime?, SC_LocalWaitTime, InterfaceAddressType, Interface+, Agent*)>

This indicates that the node element consists of all the elements between the brackets.

- <!ELEMENT Interface (Address, MaskBits)>

This indicates that the Interface element inside the node consists of all the elements between the brackets.

The following table illustrates the meaning of the operators that usually follow the element names [Ref. 7].

Operator	Description
No Operator	Must appear exactly one time
?	Must appear once or not at all
+	Must appear at least once (1 … N times)
*	May appear any number of times, or not at all (0 … N times)

Table 3.1 Recurrence Operators.

The rest of the element statements only contain the element names and their data types. For simplicity, all elements were defined as character type. This DTD does not contain attributes because the goal was to develop as simple a solution as possible. Future work can be done to add attributes to the DTD file or to create a new schema for restricting the XML files.

F. SUMMARY

XML is a new technology that is concerned with meta-data and meta-languages. It is still growing, but soon it will be the standard for data storage. It is accepted

worldwide because of its flexibility and interoperability. The technologies related to XML are growing. New parsers and new IDE's are being developed every day. The use of XML in SAAM to provide a robust and easy configuration for SAAM networks will provide a proof of XML capabilities, and opens the door for the use of XML in other aspects in the SAAM project.

The XML file was designed and correctly validated by the DTD. The IDE was very helpful in this development phase. Later, an XML file might be generated by Java programs to avoid the use of an IDE. The most important thing is to know the rules that should be enforced by the DTD files. Validating an XML file against an erroneous DTD file will produce an XML file that does not represent the situation to which we are looking for a solution.

IV. THE DEVELOPMENT OF THE JAVA PARSER

A. HOW TO ACCESS THE XML DOCUMENT THROUGH JAVA

After developing the XML and DTD files, it is now necessary to access the data stored in the XML file. In this chapter the XML document will be parsed using a parser within a Java program. The parsing process consists of some events when reading certain elements. These events are the points where the application code could be placed.

The W3C has already put specifications for XML parsing interfaces. This enables multiple vendors to provide parsers in any language they choose, but with known and easy to access methods and classes. The first two major XML parsing interfaces were DOM and SAX.

1. DOM

DOM stands for Document Object Module. It represents the XML document as a tree of objects. The hierarchical structure of the XML document, enforced by the DTD file, perfectly supports DOM. Each node will represent an object in a tree data structure. Each object can have sub-objects and there is a root object that contains all other objects. The elements are the attributes of the objects. Each object can be accessed through well-defined methods that enable the attribute access. These methods can specify the node type and data value. When the type and data are known, an application can decide what sequence of application events to execute.

The basic idea of DOM is to represent the entire XML document in a tree of objects. Of course, this tree of objects should be stored in memory. This can create a very large load on system resources for a large XML file. DOM was used to start the

development of the SAAM XML DemoStation for this thesis. However, after delving into the details, it was determined that when there was a large number of SAAM nodes, then the system ran out of memory. This memory intensive approach has led to creating a new methodology to parse the XML file and obtain the desired configuration information.

2. SAX

SAX stands for Simple API for XML. This name indicates that it is a simple application programming interface for XML. In the SAX interface, XML is parsed sequentially. Unlike DOM, which represents nodes as objects, SAX deals with single lines in the XML document. This sequential flow detects new keywords and executes certain actions based on that keyword. Some of these events, such as the start of a document, end of a document, start of a node, end of a node, start of an element and end of an element are the heart of SAX. Thus, SAX is an event driven approach to XML parsing, not an object oriented approach.

3. Why SAX?

Despite the ease of node access in DOM because of the in-memory object model, SAX is preferable to DOM because less memory is required. SAX does not require the entire document to be stored in memory, since it processes the document line by line. Thus, SAX provides a quick, less resource-intensive parsing and processing interface.

B. THE APACHE XERCES PARSER

Writing a new XML parser is not easy, and there are already many parsers available in the marketplace. IBM, SUN, Oracle, Microsoft and others all have their own parsers. The Apache project, which was named Xerces, is a complete set of parsing tools written in Java [Ref. 9]. It provides SAX, DOM and other parsing schemas. Since the

APACHE project started, most of the other XML parser vendors have agreed to support the APACHE XML parser as a kind of unofficial standard. The advantage to using Xerces is that major XML technology players like IBM and SUN support it. The use of a mature, widely accepted and standard XML parser avoids the reinvention of the wheel and allows us to concentrate on the application itself rather than the parsing.

For this reason, the Apache Xerces SAX parser was used for this project. The Apache Xerces provides the source code as well. The source code of the Apache XML SAX parser can be found in Appendix B of this thesis. After downloading the package, it was installed on the demo station and included in the Java project class path. After installing the Xerces package, it is necessary to compile the SAX parser Java file to generate the class file. At this point, a new SAX parser object can be instantiated and a new XML document can be parsed by the SAX parser.

C. THE DEMOSTATION JAVA PROGRAM

A Java program was developed for the purpose of parsing the XML configuration file. Of course, parsing cannot begin until a valid XML configuration file exists. This validity was verified by validating the XML file against a DTD file. Figure 4.1 is a UML diagram that shows the relationship between all the classes used.

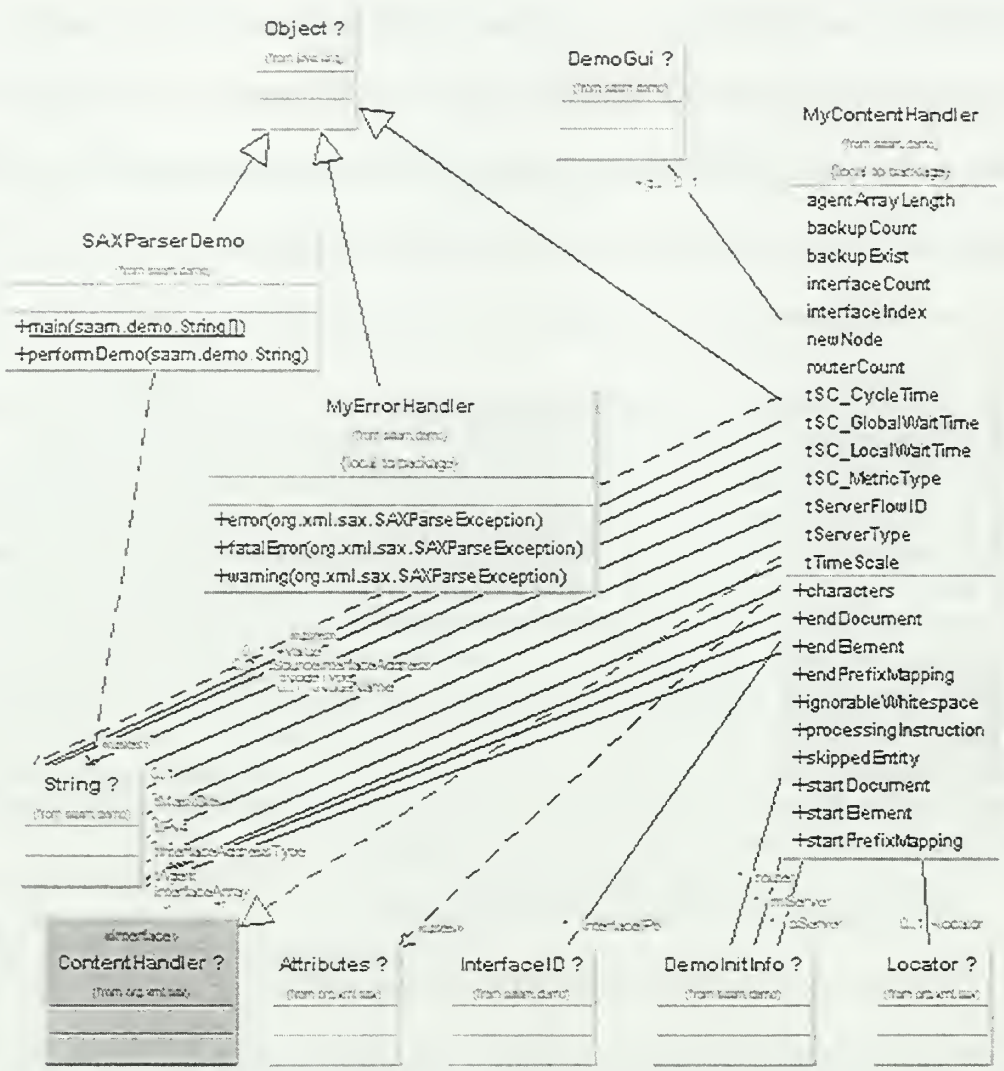


Figure 4.1. A UML Diagram of the DemoStation Program.

Three Java classes were developed for a SAAM configuration using XML: **SAXParserDemo**, **ContentHandler** and **myErrorHandler**.

1. SAXParserDemo Class: public class SAXParserDemo {}

This is the main class in the project. It starts with import statements, which import the SAAM package, some default Java classes and the org package, which contains the Xerces SAX parser classes. The purpose of this class is to start parsing the XML

document and invoke the class that will process it. Appendix A shows the entire SAXParserDemo class.

a. The Main Method: public static void main (String[] args)

(1) Purpose. This is the *main* method in the SAXParserDemo class. It displays a fileChooser window to enable the user to select the desired XML configuration file. It passes this file name after translating it to a URI (Unified Resource Identifier) to the *performDemo* method.

(2) Variables.

- guiContainer: DemoGui guiContainer = new DemoGui(""): A DemoGui object that holds the fileChooser window.
- Uri: *StringBuffer uri = new StringBuffer("file:///"):* A stringBuffer object that stores a URI format of the XML file location.
- currentFile*: File currentFile = new File("c://xml//SAAM_1.xml"):* The default file to be used.
- fileChooser: *JFileChooser fileChooser = new JFileChooser():* A fileChooser object that enables the user to select the XML file.
- result*: int result = fileChooser.showOpenDialog(guiContainer):* An integer that enables the fileChooser to appear.
- inFile: File inFile = fileChooser.getSelectedFile(): The path of the selected XML file
- parserDemo: SAXParserDemo parserDemo = new SAXParserDemo(): A SAXParser Demo object through which the performDemo method will be called.

(3) Description. The main method instantiates a GUI interface to give the user the flexibility to select files. After obtaining the normal path name of the file, that path will be converted into a URI format by appending "file:///". The use of the URI enables the SAAM configuration DemoStation to use either local files or files over the web if the URL (Unified Resource Locator) address is available.

After selecting the file, an instance of the *SAXParserDemo* object is created and the *performDemo* method is called with the URI of the XML file passed to it.

b. The PerformDemo Method: public void performDemo(String uri)

(1) Purpose. The purpose of this class is to start the parsing and processing of the XML document. It instantiates the parser and sets the XML content handler to the class developed for this purpose.

(2) Variables.

- contentHandler: *ContentHandler contentHandler = new MyContentHandler():* This object is instantiated from the ContentHandler class which causes the entire document to process evenly.
- errorHandler: *ErrorHandler errorHandler = new MyErrorHandler():* An object instantiated from the Error Handler class. It takes care of any parser error during the parsing process.
- parser: *XMLReader parser =*
- *XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser"):* An XMLReader object which is provided by the Apache Xerces parser package. A new SAX parser is instantiated from the XMLReader object.

(3) Description. This method accepts a file URI as its input and starts by creating new instances of the Content Handler and Error Handler classes. It instantiates a SAX parser object from the Xerces SAX parser class. The SAX parser is instantiated using the URI passed from the *main* method. It also sets the content handler in the SAX parser object to the contentHandler class developed for this project. It also sets the error handler to myErrorHandler class. Then, the parser method is called to parse the file with the associated URI.

2. MyContentHandler Class: class MyContentHandler implements ContentHandler {}

This is the processing class in the project. It implements the Apache Xerces ContentHandler class. The source code for this class is provided in Appendix C. The methods in this class take care of certain events that happen during the sequential reading of the document. Events like reaching the beginning or end of a document, node or

elements specify which actions are to be executed when these events occur. The content handler handles these events by extracting the data from the XML file and then executing the correct configuration action, such as creating nodes and interfaces or sending agents. All SAAM configuration specific processing is done here in the content handler.

The class variables and the major methods in the content handler are discussed below. The source code for MyContentHandler.java is in Appendix A.

a. The Class Variables

- gui: *public DemoGui gui = new DemoGui(""):* A DemoGui object that displays the configuration messages and the status of the XML file parsing.
- locator: *private Locator:* keeps track of the current location inside the XML document.
- element: *public String element = "":* A variable that stores the name of the current element.
- value: *public String value = "":* A variable that stores the value of the current element.
- Temporary variables to store values read from the XML file.
    - tIPv4: *String tIPv4 ="":* A temp variable that stores the value of the IPv4 element.
    - tSourceInterfaceAddress: *String tSourceInterfaceAddress "":* A temp variable that stores the value of the Address element.
    - tInterfaceAddressType: *String tInterfaceAddressType "":* A temp variable that stores the value of the InterfaceAddressType element.
    - tMaskBits: *String tMaskBits = "":* A temp variable that stores the value of the MaskBits element.
    - tAgent: *String tAgent ="":* A temp variable that stores the value of the Agent element.
    - tNodeName: *String tNodeName ="":* A temp variable that stores the value of the NodeName element.
    - tNodeType: *String tNodeType ="":* A temp variable that stores the value of the NodeType element.
    - tServerType: *int tServerType =5:* A temp variable that stores the type of server. If the NodeType element is a Primary Server then

this value is "0". If the NodeType element is a Backup Server then this value is "1".

- tServerFlowID: *int , tServerFlowID =5:* A temp variable that stores the value of the ServerFlowID element.
- tTimeScale: *int tTimeScale =5:* A temp variable that stores the value of the TimeScale element.
- tSC_MetricType: *int tSC_MetricType =5:* A temp variable that stores the value of the SC_MetricType element.
- tSC_CycleTime: *int tSC_CycleTime =5:* A temp variable that stores the value of the SC_CycleTime element.
- tSC_GlobalWaitTime: *int tSC_GlobalWaitTime =5:* A temp variable that stores the value of the SC_GlobalWaitTime element.
- tSC_LocalWaitTime: *int tSC_LocalWaitTime =5:* A temp variable that stores the value of the SC_LocalWaitTime element.

- interfaceArray: *String [] [] interfaceArray = new String [100][5]:*An array that holds all the nodes interfaces. It contains the NodeType, NodeName, NodeTypeNumber, SourceIP and DestIP.
- agentArray: *String [] [] agentArray = new String [100][2]:* An array of agent objects. The IPv4 and the Agent name in each node are stored in this array.
- interfaceCount: *int interfaceCount = 0:* A counter that tracks the total number of interfaces in the interface array.
- interfaceIndex: *int interfaceIndex = 0:* An index that walks through the interfaceArray.
- agentArrayLength: *int agentArrayLength = 0:* An index that walks through the agent array.
- Arrays to hold node objects:
  - router: *DemoInitInfo[] router = new DemoInitInfo[100]:* An array of router objects. It can handle up to 100 routers.
  - bServer: *DemoInitInfo[] bServer = new DemoInitInfo[10]:* An array of backup server objects. It can handle up to 10 backup servers.
  - mServer: *DemoInitInfo[] mServer = new DemoInitInfo[1]:* An array of server objects. It handles only one primary server.
- interfaceIP6: *InterfaceID [] interfaceIP6 = new InterfaceID[100]:* An array of InterfaceIP6 objects. It can handle up to 100 interfaces.

- routerCount: *int routerCount = 0:* A counter for the number of routers in the SAAM configuration.
- backupCount: *int backupCount = 0:* A counter for the number of backup servers in the SAAM configuration
- backupExist: *boolean backupExist = false:* A Boolean variable that indicates if there is a backup server or not.
- newNode: *boolean newNode = false:* A Boolean variable that indicates if this is a new node or not.

b. ***The startDocument Method: public void startDocument() throws SAXException***

(1) Purpose. This method can contain any statement to be executed upon opening an XML document.

(2) Variables. None.

(3) Description. This method only sends a message to the output window to indicate the beginning of the parsing process.

c. ***The startElement Method: public void startElement(String namespaceURI, String localName, String rawName, Attributes atts) throws SAXException {***

(1) Purpose. This method represents the event of reaching a new element. It is important to track the names of the elements.

(2) Variables.

- **LocalName:** ***String localName:*** A string that stores the name of the current element
- **NamespaceURI:** ***String namespaceURI:*** A string that stores the name space if it were used.
- **RawName:** ***String rawName:*** A string that stores the raw name if it were used.
- **Atts:** ***Attributes atts:*** An attribute object that stores the current element attributes if they were used in the DTD file.

(3) Description. When a new element is processed, a message is sent to the output window. The name of the element "localname" is assigned to the temp variable "element".

d. The Characters Method: public void characters(char[] ch, int start, int end)

(1) Purpose. The main purpose of this method is to retrieve the element's value and make it available to the application.

(2) Variables

- ch: *char[] ch:* An array of characters to hold the value of the element.
- start: *int start:* An integer that indicates the start of the value string.
- end: *int end:* An integer that indicates the end of the value string.

(3) Description. When a new set of characters is detected, these characters are read into an array of characters. This array of characters is then assigned to the temp application variable "value".

e. The endElement Method: public void endElement(String namespaceURI, String localName, String rawName) throws SAXException

(1) Purpose. When the end of an element is reached, a certain action should occur. Mainly, this action will store the element's value in its corresponding temporary variable to be used to create an object with a specific configuration. Also, node, interface and agent arrays are built into this method. After gathering all the data necessary to create a node, that node can be created by instantiating a DemoInitInfo object.

(2) Variables. This method uses the public variables declared in the class.

(3) Description. This method always checks the name of the current element. If the name is "Node" then it knows that this is a new node. If the NodeType value is "Primary" then the server type is "0", otherwise it is "1". At each end of an element, it checks the name and stores the value of the element in the temporary variable created to hold that value.

(4) If the element is MaskBits then it knows that there is a new interface and the interface data should be collected in the interface array. If the element is Agent then a new agent array entry is added to the agent array.

(5) If the end of a node is detected, then it is time to create that node. This is the difference between SAX and DOM. It is not necessary to wait until the entire document is read. The processing is done node by node. The node type is checked in order to create a DemoInitInfo object with the correct set of configuration attributes.

f. The EndDocument Method: public void endDocument() throws SAXException

(1) Purpose. This method finalizes the entire configuration process. As indicated by its name, this method will execute actions when the end of the XML document is detected. In our case, it is necessary to activate the nodes created, connect the interfaces and send each agent to its node.

(2) Variables.

- sourceIP: *String sourceIP =""*: A temporary variable to store the source interface address.
- destIP: *destIP = ""*: A temporary variable to store the destination interface address.

(3) Description. This method goes through the interface array and starts creating the interfaces that belong to each node object in the test bed. Calling

the addNewInterface method from the DemoInitInfo class creates the interfaces. For each interface created, a lookup for the correspondent interface on a neighboring node is carried out. If the network addresses of two interfaces are equal then the interfaces are considered to be on the same network and they should be directly linked. Each interface is connected to the corresponding interface on the next node using the **isConnectedTo** method. Also, all nodes are activated here by going through the arrays that hold similar nodes. Finally, all agents in the agent array are sent to their owner nodes. The DemoInitInfo's sendAgent method is called to send the agent.

This method concludes the configuration process. If there were no errors or exceptions, a SAAM test bed with the desired configuration is in place.

3. MyErrorHandler: class MyErrorHandler implements ErrorHandler

MyErrorHandler implements the Apache SAX ErrorHandler interface and defines the behavior of the SAX content handler events associated with an XML document's errors. The source code of MyErrorHandler.java is found in Appendix A, while the Apache SAX ErrorHandler source code can be found in Appendix D. This method reports any exceptions or errors that are related to the parsing process.

a. The Warning Method: public void warning(SAXParseException exception) throws SAXException

This method will report a warning if a non-critical parsing error has occurred. This warning indicates that while no XML rules were broken, something appears to be incorrect or missing.

b. The Error Method: public void error(SAXParseException exception) throws SAXException

This method will report an error if a non-critical parsing error has occurred. This error indicates that even if an XML rule were broken, the parsing can continue.

c. The fatalError Method: public void fatalError(SAXParseException exception) throws SAXException

This method will report a fatal error if a critical parsing error has occurred. This fatal error indicates that the parsing process cannot be continued because of a major violation to XML rules.

D. INTEGRATION AND TESTING

1. Integration

After developing the XML and the Java portion of this configuration project, the next step is to integrate the solution with the existing SAAM project. The **SAXParserDemo** program has the flexibility to accept any path for the XML file. A Java Swing FileChooser GUI object provides this flexibility. Figure 4.2 illustrates this GUI interface.

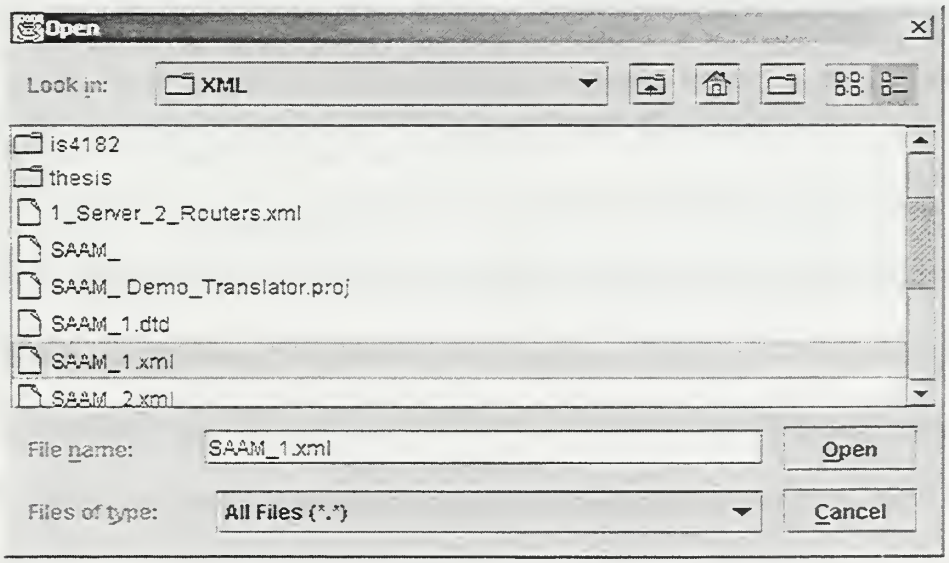


Figure 4.2. The FileChooser Object.

The **SAXParserDemo** program was integrated into the SAAM project by placing source and class files in the SAAM\Demo directory.

The Apache Xerces parser package was included in the project class path. The "org" folder that contains all the parsing classes was located in the SAAM root directory. The **SAXParserDemo** program imports all the needed classes from this folder.

2. Testing

The developed XML, DTD files and the Java program were tested. The Java program starts by asking for the location of the XML file which should be validated against a DTD file. The Java parsing program successfully reads the contents of the XML, extracts the needed configuration information and sets up a network topology for the SAAM network. After testing the configuration on each newly configured node, satisfactory results were achieved.

In our evaluation for the solution that we have provided, we have taken into consideration some testing criterion.

a. Ease of use

The developed XML approach to configure SAAM network was a lot easier to use than the previous configuration approaches. It was very difficult to create a new topology using the hard-coded configuration information inside the Demo Station program. On the other hand, the ASCII approach was easier, but not that much. The user has to spend a significant amount of time learning the syntax. In addition, there was no tool to make sure that the manually entered commands and values are correct. The XML solution provided an easy configuration approach relieving much of error-prone task from the user and automating as much as possible of the configuration process.

b. User Interface

The XML solution provided a friendly user interface. The user can build the topology or expand it very easily through a GUI interface. The tool used to create the XML files was very powerful in displaying the data in different views that makes building large topologies an easy task. Appendix G provides an easy guide to use the tool. Also, the user has the choice to build his topology either from a local file or a location on the Internet through an easy to use File Chooser GUI object. During the parsing process the user stays in contact with the program by displaying debugging messages. The user can be sure that the program has established the topology correctly by checking the last set of debugging messages indicating the success of sending all the configuration information to its corresponding node. The figure below shows the results of the parsing.

c. Portability

Even though we didn't test the new Demo Station under different platforms, we were confident that it would work because of the portability of the XML and Java technologies. Unlike the ASCII approach that might not work on certain platforms, the XML can be used on any platform

d. Performance

The speed of parsing of the XML configuration file was almost the same as the previous approaches. The Java-Parser program worked as anticipated without any problems. Unlike the previous configuration approaches, the XML saved some time at the early steps of the topology creation through the XML IDE.

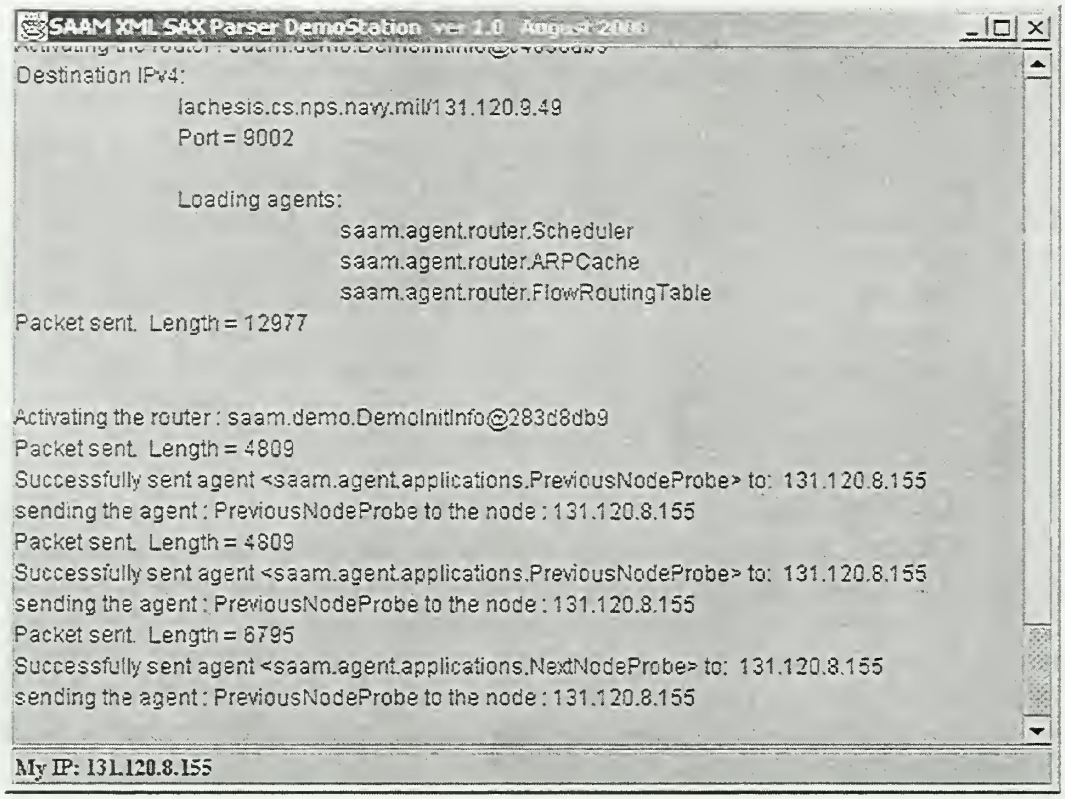


Figure 4.3. The Final Results from the Parsing Program.

E. SUMMARY

The design of the XML Java parser was a success. We could read our configuration information from the XML files successfully. The solution was integrated into the SAAM system and tested many times to make sure everything is going well. The most important issue is to validate the XML file. If the user is parsing an invalid file the parsing process might fail. The XML tool that we were using has a powerful feature for checking validity, which helps a lot.

V. THE CONCLUSION AND FUTURE WORK

A. CONCLUSIONS

The implementation of the XML approach to configuring the SAAM network was a success. It provided an easy, flexible and efficient way to configure a set of SAAM servers and routers. The XML IDE was easy to use and user friendly. By contrast, using the hard-coded approach proved to be inefficient and a waste of time when configuring new topologies in SAAM. Although the use of the ASCII configuration approach was an improvement, it was nevertheless a platform-dependent solution without any tools available to check the validity of the configuration information before starting the configuration-parsing program.

The use of XML provides a platform-independent, worldwide standard, robust, flexible and interoperable solution for the SAAM network configuration. The use of a Java-based, open-source and widely supported XML parser contributed to the ease in developing the Java programs. Many major XML technology providers support the Apache parser project.

B. LESSONS LEARNED

Several lessons, in addition to the proof that XML can function very well as a configuration language, were learned during the development of this thesis.

1. SAX vs. DOM

After starting the development of the Java program using the DOM parser, it became apparent that it was a memory intensive approach because the entire XML document is stored in memory. Even though DOM is attractive because of the ease in moving up and down the hierarchy, a developer cannot afford to give up the large amount

of memory needed for processing purposes. SAX is an interpretative parsing technique that is less memory intensive. The processing of XML content occurs while it is being read, which makes it a more memory-efficient parser.

2. XML IDE

Using an XML IDE was very helpful. It provided an easy to use GUI interface with which to build the XML configuration file. It makes it possible to switch between a tabular view of the topology and the native textual XML files. By using the tabular view, a new topology could be established within minutes with the guarantee that this configuration would be valid. It is also possible to generate the DTD file based on the XML file. This allows the user to write the XML file once, and then generate the rules for the DTD file instead of spending a lot of time thinking about the rules that should be supported by the DTD file.

3. XML in Networking

XML is having an increasing impact on every aspect of Information Technology. It enables the development of network management systems that can overcome hardware and software incompatibility issues. It also facilitates the development of a smart, easy to use, efficient and unified approach to network configuration.

C. FUTURE WORK

This thesis was the first to be written on the use of XML with network configurations. It proved that XML is a viable approach for configuring networks and network devices. It is hard to imagine how the world would be if XML were supported by every network device. Network management systems from any vendor could talk to any network device regardless of the manufacturer and the operating system being used.

This is critical for a system like SAAM, which has universal QoS as its main goal. XML is the only approach that can provide such interoperability between heterogeneous systems and networks. Currently there is no other approach that could contribute to interoperability more than what XML could.

Many aspects can be studied now after the use of XML as a network configuration language has been proven. Some of these aspects are discussed below.

1. The Implementation of the PIB

The first implementation of the Path Information Base in SAAM was built as an Oracle relational database. Due to large access times, this design was abandoned and a new object oriented design was developed. XML is an alternative for the PIB current implementation, which might be more efficient. The XML document can be easily converted into a hierarchical tree of objects. This XML data can be easily read into memory using XML parsing modules, which are developed without difficulty using Java. This is an important aspect of SAAM that should be investigated.

2. The Use of XML to Exchange Messages (Packets)

The packet exchange between nodes using XML should also be investigated. XML is a general data format that is widely used in data exchange. A more advanced study can be conducted to decide whether the use of XML for exchanging packets between SAAM nodes will have a positive impact on the system.

3. The Development of a Web-Based Network Management System

XML makes it possible for heterogeneous network devices to talk to each other. If XML is supported on every network device then a robust web-based network management system can be developed that can access configuration parameters and

performance metrics of all network nodes. A web browser can send the configuration information to the network nodes, observe the status of each node and provide up-to-date information on network performance. This centric network view and management can lead to better QoS networks.

4. The Development of Smart Self-Configurable Nodes

After the initial network configuration, if any node goes down, then the whole topology has to be restarted. A new Java model can be built to run on each node that stores the configuration information received from the DemoStation in XML files. When that specific node is interrupted, it could rebuild its configuration information and report its status to the regional server and neighboring routers.

In summary, the flexibility of XML provides several interesting research opportunities for use in network management applications.

APPENDIX A. XML SAX PARSER DEMO STATION SOURCE CODE

```
/*
 * Program : SAXParserDemo.java
 * Updated : Sep 6, 2000
 * Author  : Mohammad Ababneh
 * Purpose : This Demo Station Configuration program will setup the topology
 *           for the SAAM network. It is reading the configuration
 *           information from an XML file that can be selected by the user.
 *           The XML file contains a special configuration meta-data
 *           developed for this purpose.
 *           There is no need to hardcode the configuration information inside
 *            the demo station programs.
 */

 //10 August 00 [Mohammad Ababneh] created


package saam.demo;

// saam demo imports
import saam.*;
import saam.control.*;
import saam.message.*;
import saam.agent.*;
import saam.router.*;


import saam.net.*;
import saam.util.*;
import saam.demo.DemoInitInfo;

import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;

// Apache Xerces XML-SAX parser imports
import java.io.IOException;

import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

/**
 * Class  : SAXParserDemo
 * Purpose: will take an XML file and parse it using SAX. the XML file contains
the configuration
 *           information of the SAAM servers and routers.
 *
 */
public class SAXParserDemo {

   /**
    * Method     : performDemo
    * Purpose    : This method parses the file, using registered SAX handlers,
and output
    *              the events in the parsing process cycle.
    * Parameters : uri  ( String uri ) URI of the file to parse.
    */

   public void performDemo(String uri) {
      System.out.println("Parsing XML File: " + uri + "\n\n");

      // Get instances of the major classes needed
      ContentHandler contentHandler = new MyContentHandler();
```

```
      ErrorHandler   errorHandler   = new MyErrorHandler();

      try {
         // Instantiate a SAX parser
         XMLReader parser = XMLReaderFactory.createXMLReader(
                    "org.apache.xerces.parsers.SAXParser");

         // Register the content handler
         parser.setContentHandler(contentHandler);

         // Register the error handler
         parser.setErrorHandler(errorHandler);

         // Parse the document in the location uri
         // it can be local or in a URL
         parser.parse(uri);

      } catch (IOException e) {
          System.out.println("Error reading URI: " + e.getMessage());
      } catch (SAXException e) {
           System.out.println("Error in parsing: " + e.getMessage());
      }


   }  // end performDemo

   /**
    * Method     : main
    * Purpose    : The main method in the SAXParserDemo class.
    * Parameters : String[] args
    */

    public static void main(String[] args) {

       DemoGui guiContainer = new DemoGui("SAAM XML SAX Parser DemoStation  ver
1.0   August 2000 ");
        StringBuffer uri = new StringBuffer("file:///");


       //display the File Chooser here
           // open the file through a file chooser object and read data

        // default directory in File Chooser to be the directory
        File currentFile = new File("c://xml//SAAM_1.xml");
       JFileChooser fileChooser = new JFileChooser();
       fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
       fileChooser.setCurrentDirectory(currentFile);

       // display the dialog box in the screen center
       int result = fileChooser.showOpenDialog(guiContainer);
       File inFile = fileChooser.getSelectedFile();

       // format the location of the XML file as an URI by adding the path to
"file:///"
       uri.append(inFile.getPath());

           System.out.println(" URI of the XML file is : " + uri);


        // instantiate a new SAXParserDemo object
           SAXParserDemo parserDemo = new SAXParserDemo();

           //  call  the  performDemo  method  that  will  call  the  XML  content
handler
       parserDemo.performDemo(uri.toString());

        try{
          Thread.sleep(3000000); //sleep 30 seconds
       }
       catch(InterruptedException ie){
```

```
                //gui.sendText("problem after initrouter thread sleep");
             System.out.println("problem after initrouter thread sleep");
          }

           System.exit(0);

      }  // end main method

}     // end SAXParserDemo



/**
 * Class  : MyContentHandler
 * Purpose: implements the SAX interface and parses the document
 *          based on the events that happen inside the documnet
 *          it establishes the basic data structires that gather the
 *          configuration information from the XML file.
 */
class MyContentHandler implements ContentHandler {

   public DemoGui gui = new DemoGui("SAAM XML SAX Parser DemoStation  ver 1.0
August 2000 ");

   /** Hold onto the locator for location information */
   private Locator locator;

   /*************************/
   // all global variables
   // temp variables to store values read from the XML file

   public String element = "", value ="";  // element & value

   String   tIPv4   ="",   tSourceInterfaceAddress   ="",tInterfaceAddressType,
tMaskBits ="", tAgent ="";
   String tNodeName ="", tNodeType ="";
   int tServerType =5, tServerFlowID =5,tTimeScale =5;
   int   tSC_MetricType   =5,   tSC_CycleTime   =5   ,   tSC_GlobalWaitTime   =5,
tSC_LocalWaitTime =5;


   // array of interface objects: node_type, node_name, node_type_number
   //                             source_IP, dest_IP, maskBits
   String [] []  interfaceArray = new String [100][6];
   String [] []  agentArray = new String [100][2];  // array of agent objects


      // IPv4 , Agent name
   int interfaceCount   = 0;                      //   the   total   number   of
interfaces in the interface array
   int interfaceIndex   = 0;                       // interfaceArray index
   int agentArrayLength = 0;                          // index of agents in
the agent array

   DemoInitInfo[] router  = new DemoInitInfo[100];  // array of router objects
   DemoInitInfo[] bServer = new DemoInitInfo[10];   // array of backup objects
   DemoInitInfo[] mServer = new DemoInitInfo[1];    // array of server objects

   InterfaceID [] interfaceIP6  = new InterfaceID[100]; // array of InterfaceIP6
objects

   int routerCount = 0;                           // router sequential counter
   int backupCount = 0;                           //  backup  sequential  server
counter

   boolean backupExist = false;  // variable to decide if a backup server exists
   boolean newNode = false;      // variable to decide if this is a new node
```

```
   /*********************************************************************/


    /**
     * Method      : setDocumentLocator
     * Purpose      : This method gives the capability to define the exact
location while
     *          parsing the XML file
     * Parameters : Locator locator
     */
   public void setDocumentLocator(Locator locator) {

       System.out.println("    * setDocumentLocator() called");

      // We save this for later use if desired.

       this.locator = locator;
   }

    /**
     * Method      : startDocumnet
     * Purpose     : In this method there can be any kind of statements
     *          that we would like to occur when we first open an XML document
* Parameters : String[] args
     * Parameters :
     * throws      : SAXException when things go wrong
     */
    public void startDocument() throws SAXException {

        System.out.println("Parsing begins...");


    }

    /**
     * Method      : endDocumnet
     * Purpose     : This method indicates the end of the XML document.
     *          It is reached when it finishes all the parsing events insidethat
we would like to occur when we first open an XML document     * Parameters :
String[] args
     * Parameters : None
     * throws      : SAXException when things go wrong
     */
    public void endDocument() throws SAXException {

        System.out.println("...Parsing ends.");


gui.sendText("================================================================");
       gui.sendText("Start creating interfaces when reaching the end of the
document");

gui.sendText("================================================================");


       int routerCount = 0;  // counter for router in interface array
       int backupCount = 0;  // counter for backup servers in interface array

       // start creating interfaces on the nodes

       for (int idx = 0 ; idx < interfaceIndex ; idx++) {

          if (interfaceArray[idx][0].startsWith("P")) {
             interfaceIP6[idx]                                                =
mServer[0].getNewInterface(interfaceArray[idx][2]);
             gui.sendText("creating primary server interface number :     " +
idx);
             System.out.println("creating server interface number :    " + idx);
          }
          else if (interfaceArray[idx][0].startsWith("B")) {
```

```
            backupCount = Integer.parseInt(interfaceArray[idx][1]);
            interfaceIP6[idx]                                                    =
bServer[backupCount].getNewInterface(interfaceArray[idx][2]);
            gui.sendText("creating backup server interface number :   " + idx);
            System.out.println("creating backup server interface number :   " +
idx);
         }
         else {
            routerCount = Integer.parseInt(interfaceArray[idx][1]);
            interfaceIP6[idx]                                                    =
router[routerCount].getNewInterface(interfaceArray[idx][2]);
            gui.sendText("creating router interface number :   " + idx);

         }  // end else

         interfaceCount = idx;   // number of interfaces in the array

      } // end for

      // find which interfaces are on the same network and sharing the same
link
      // by finding the network address and comparing the two interface
addresses

      for ( int i=0; i<=interfaceCount; i++){

         for ( int j=i+1; j<=interfaceCount; j++){

            if
((interfaceIP6[i].getIPv6().getNetworkAddress()).equals(interfaceIP6[j].getIPv6(
).getNetworkAddress())){
               gui.sendText("--- equal interface 1"+ interfaceIP6[i].getIPv6()
+"  interfacse 2" + interfaceIP6[j].getIPv6());

               // find the destination IPv6 address for the source IPv6 address
               interfaceArray[i][3] = interfaceIP6[j].getIPv6().toString();
               interfaceArray[j][3] = interfaceIP6[i].getIPv6().toString();

            }
            else {

               // gui.sendText(" not equal   " + interfaceIP6[i].getIPv6()+"
interfacse 2" + interfaceIP6[j].getIPv6());
            }  // end if

         }    // end for j

      }       // end for i


      // display the interface and agent arrays

      gui.sendText("
//////////////////////////////////////////////////////////////////////");
      gui.sendText(" // display the interface array.... which has   " +
interfaceIndex + "  interfaces");

      for (int idx = 0; idx < interfaceIndex; idx++) {
         gui.sendText(interfaceArray[idx][0]+ " " + interfaceArray[idx][1] +
"   "   +interfaceArray[idx][2]   +   "   "   +interfaceArray[idx][3]   +   "   "
+interfaceArray[idx][4]);
      }
      gui.sendText("//  display  the  agent  array....  which  has    " +
agentArrayLength + "  agents");

      for (int idx = 0; idx < agentArrayLength; idx++) {

         gui.sendText(agentArray[idx][0]+ " " + agentArray[idx][1] );
      }
```

```
        //************************************************************/
        // connecting the Interfaces
        String sourceIP, destIP = "";  // source IPv6 and destination IPv6

        for (int sourceCount=0; sourceCount <= interfaceCount; sourceCount++) {
           sourceIP = interfaceArray[sourceCount][2];

           // look for the corresponding interface to link with
           for (int destCount=0; destCount <= interfaceCount; destCount++) {
              destIP = interfaceArray[destCount][3]; // was [2] - crc

              if (sourceIP.equals(destIP)) {
                 if (interfaceArray[sourceCount][0].startsWith("P")) {       //
check the node type
                    // call the connect method
                    mServer[0].isConnectedTo((interfaceIP6[destCount]));
                 }
                 else if (interfaceArray[sourceCount][0].startsWith("B")) { //
check the node type
                    int                          backupIndex                          =
Integer.parseInt(interfaceArray[sourceCount][1]);
                    // call the connect method

bServer[backupIndex].isConnectedTo((interfaceIP6[destCount]));
                 }
                 else if (interfaceArray[sourceCount][0].startsWith("R")) { //
check the node type
                    int                          routerIndex                          =
Integer.parseInt(interfaceArray[sourceCount][1]);
                    //call the connect method

router[routerIndex].isConnectedTo((interfaceIP6[destCount]));
                 }

                 gui.sendText("connecting      node      :             "      +
interfaceArray[sourceCount][0] + " " + interfaceArray[sourceCount][4] +" with "
+ interfaceArray[destCount][3]);

              } // end if

           }    // end for

        }       // end for


        // Activating the nodes
        // use a loop to activate server, backup servers and routers

         // activate the server
        mServer[0].activate();
        gui.sendText("Activating the Server" );

         // activate the routers
        for (int idx = 0; idx <= routerCount; idx++) {
           router[idx].activate();
           gui.sendText("Activating the router : " + router[idx] );
        }

         // activate the backup servers
        if (backupExist) {
           for (int idx = 0; idx <= backupCount; idx++) {
              bServer[idx].activate();
           }

           backupExist = false;  // there is no backup servers
        }


        // sending the agents to the nodes
```

```
        StringBuffer agentStringBuffer = new StringBuffer("");

        for (int agentCounter = 0; agentCounter < agentArrayLength;
agentCounter++) {

             agentStringBuffer.append("saam.agent.applications.");       //  the
place where application agents are
            agentStringBuffer.append(agentArray[agentCounter][1]);     //  agent
name


DemoInitInfo.sendAgent(gui,agentArray[0][0],agentStringBuffer.toString());

            gui.sendText("sending the agent : " + agentArray[0][1] + " to the
node : " + agentArray[0][0] );
            agentStringBuffer.delete(0,agentStringBuffer.length()); // clear the
string buffer to hold a new agent

        } // end for

   }   // end of endDocument Method

   /**
    * Method     : processingInstruction
    * Purpose    : This method is used when processing instructions are found
    *          in the XML file. The current XML file doesn't support PI
    * Parameters : String target, String data
    * throws     : SAXException when things go wrong
    */
   public void processingInstruction(String target, String data)
      throws SAXException {
      System.out.println("PI: Target:" + target + " and Data:" + data);
   }

   /**
    * Method     : startPrefixMapping
    * Purpose    : This method is used in case XML name spaces are used.
    *               In the current use of XML. name spaces are not supported.
    * Parameters : String prefix, String uri
    * throws     : SAXException when things go wrong
    */
   public void startPrefixMapping(String prefix, String uri) {
      System.out.println("Mapping starts for prefix " + prefix +
                         " mapped to URI " + uri);
   }

   /**
    * Method     : endPrefixMapping
    * Purpose    : This method is used in case XML name spaces are used.
    *               In the current use of XML. name spaces are not supported.
    * Parameters : String prefix, String uri
    * throws     : SAXException when things go wrong
    */

   public void endPrefixMapping(String prefix) {
      System.out.println("Mapping ends for prefix " + prefix);
   }

   /**
    * Method     : startElement
    * Purpose     : This reports the occurrence of an actual element.  It will
include
    *               the element's attributes, with the exception of XML
vocabulary
    *             specific attributes like "DTD",....
    * Parameters : String namespaceURI, String localName,
    *                        String rawName, Attributes atts)
    * throws     : SAXException when things go wrong
    */
```

```
   public void startElement(String namespaceURI, String localName,
                            String rawName, Attributes atts)
      throws SAXException {

      System.out.print(" \n startElement: " + localName);

      element = localName;

   }


   /**
    * Method     : endElement
    * Purpose    : Indicates the end of an element is reached.  Note that
    *              the parser does not distinguish between empty elements and
    *                non-empty elements, so this will occur uniformly.
    *                We gather the value of the element when we reach the end
    *                of the element
    * Parameters : String namespaceURI, String localName, String rawName
    * throws     : SAXException when things go wrong
    */
   public void endElement(String namespaceURI, String localName,
                          String rawName)
      throws SAXException {

      System.out.println("\n element = " + element + "   value =  " + value);


      if (element.equals("NODE")) {

         newNode  = true;    // this is a new node

         gui.sendText("==============   The   beginning   of   a   new   node
==================");
      }
      else if   (element.equals("NodeType")) {

         tNodeType = value;

         if (tNodeType.startsWith("P")) {
            tServerType=0;
         }
         else {
            tServerType=1;
         }

         gui.sendText("==============   The   beginning   of   a   new   node
==================");
         gui.sendText(" Node Type : " + tNodeType);
      }

      else if   (element.equals("NodeName")) {
         tNodeName = value;
         gui.sendText(" Node Name : " + tNodeName);
      }

      else if   (element.equals("IPv4")) {
         tIPv4 = value;
         gui.sendText(" IPv4 : " + tIPv4);
      }

      else if (element.equals("ServerFlowID")) {
         tServerFlowID = Integer.parseInt(value);
         gui.sendText(" ServerFlowID : " + tServerFlowID);
      }

      else if (element.equals("TimeScale")) {
         tTimeScale = Integer.parseInt(value);
         gui.sendText(" TimeScale : " + tTimeScale);
      }
```

```
      else if (element.equals("SC_MetricType")) {
         tSC_MetricType = Integer.parseInt(value);
         gui.sendText(" SC_MetricType : " + tSC_MetricType);
      }

      else if (element.equals("SC_CycleTime")) {
         tSC_CycleTime = Integer.parseInt(value);
         gui.sendText(" SC_CycleTime : " + tSC_CycleTime);
      }

      else if (element.equals("SC_GlobalWaitTime")) {
         tSC_GlobalWaitTime = Integer.parseInt(value);
         gui.sendText(" SC_GlobalWaitTime : " + tSC_GlobalWaitTime);
      }

      else if (element.equals("SC_LocalWaitTime")) {
         tSC_LocalWaitTime = Integer.parseInt(value);
         gui.sendText(" SC_LocalWaitTime : " + tSC_LocalWaitTime);
      }

      else if (element.equals("InterfaceAddressType")) {
         tInterfaceAddressType = value.toString();
         gui.sendText(" InterfaceAddressType : " + tInterfaceAddressType);
      }

      else if (element.equals("Interface")) {

         boolean newInterface = true;
         gui.sendText(" A new interface was descovered");
      }

      else if (element.equals("Address")) {
         tSourceInterfaceAddress = value.toString();
         gui.sendText(" SourceInterfaceAddress : " + tSourceInterfaceAddress);
      }

      else if (element.equals("MaskBits")) {
         tMaskBits = value.toString();
         gui.sendText(" MaskBits : " + tMaskBits);
      }

      // at the end of Interface element, add an entry to the Interface array
      if (localName.equals("MaskBits")) {
         gui.sendText(" Add a row to the interface matrix");

         // fill the interface address array
         interfaceArray[interfaceIndex][0] = tNodeType;
         interfaceArray[interfaceIndex][4] = tNodeName;
         interfaceArray[interfaceIndex][2] = tSourceInterfaceAddress;
         interfaceArray[interfaceIndex][5] = tMaskBits;


         if (interfaceArray[interfaceIndex][0].startsWith("R")) {      // router
interface
            String str = Integer.toString(routerCount);
            interfaceArray[interfaceIndex][1] = str;
         }
         else  if  (interfaceArray[interfaceIndex][0].startsWith("B"))  {    //
backup server interface
            String str = Integer.toString(backupCount);
            interfaceArray[interfaceIndex][1] = str;
         }

         interfaceIndex++;    // increase number of interfaces

      }   //end if (localName.equals("Interface")) {

      else if (element.equals("Agent")) {
         tAgent = value.toString();
```

```
         gui.sendText(" Agent : " + tAgent);
      }

      // at the end of Agent element, add an entry to the Agent array
      if (localName.equals("Agent")) {

         gui.sendText(" Add the agent to the agent array ... if you want to
check display the array");
         agentArray[agentArrayLength][0] = tIPv4;  //node IP Address;
         agentArray[agentArrayLength][1] = tAgent; // agent name
         agentArrayLength++;   // increase number of agents
      }



      //    start creating the node once all data is collected
      //            When finding the end of the node </node>

      try{
          gui.setTextField("My                          IP:                          "+
InetAddress.getLocalHost().getHostAddress());

         if (localName.equals("Node")) {

         gui.sendText("==============    The    end    of    a    the    node
==================");

         // display the arrays contents
         gui.sendText(" // display the interface array.... which has   " +
interfaceIndex + "  interfaces");
         for (int idx = 0; idx < interfaceIndex; idx++) {
            gui.sendText(interfaceArray[idx][0]+ " " + interfaceArray[idx][1] +
"   "   +interfaceArray[idx][2]   +   "   "   +interfaceArray[idx][3]   +   "   "
+interfaceArray[idx][4]);
         }
         gui.sendText("//  display  the  agent  array....  which  has    "  +
agentArrayLength + "  agents");
         for (int idx = 0; idx < agentArrayLength; idx++) {

            gui.sendText(agentArray[idx][0]+ " " + agentArray[idx][1] );
         }

          if (tNodeType.startsWith("P")) {
              // creating a primary server node
            Configuration cfMainServer = new Configuration(
                                  (byte)tServerType,
                                  tServerFlowID,
                                  (byte)tSC_MetricType,
                                  tSC_CycleTime,
                                  tSC_GlobalWaitTime);
            mServer[0]                    =                                  new
DemoInitInfo(gui,InetAddress.getByName(tIPv4),tTimeScale,cfMainServer);

            gui.sendText("  the  server  is  created  ,  and  its  IP  is  :    "  +
InetAddress.getByName(tIPv4));
         }
         else if (tNodeType.startsWith("B")){
            // creating a backup server node for as many backup server as there
are
            backupExist = true;
            Configuration cfBackupServer = new Configuration(
                           (byte)tServerType,
                           tServerFlowID,
                           (byte)tSC_MetricType,
                                 tSC_CycleTime,
                                 tSC_GlobalWaitTime);
            bServer[backupCount]                          =                    new
DemoInitInfo(gui,InetAddress.getByName(tIPv4),tTimeScale,cfBackupServer);
```

```
            gui.sendText(" the backup server is created , and its IP is :   " +
InetAddress.getByName(tIPv4));
            backupCount++;
          }
         else {
             // creating a router node
             router[routerCount]                =                                  new
DemoInitInfo(gui,InetAddress.getByName(tIPv4),tTimeScale);
             gui.sendText(" a new router is created , and its IP is :   " +
InetAddress.getByName(tIPv4));
             routerCount++;

         }      // end router

      } // end if (localName.equalus(Node)
      }
      catch(UnknownHostException uhe){
          gui.sendText(uhe.toString());

      }


    }  // end endElement method

   /**
    * Method     : characters
    * Purpose    : This method returns the real value stored in the XML element
    *              It is then converted to a string " array of characters
    * Parameters : char[] ch, int start, int end
    * throws     : SAXException when things go wrong
    */

    public void characters(char[] ch, int start, int end)
      throws SAXException {

      String s = new String(ch, start, end);
      value = s;
   }

   /**
    * Method     : characters
    * Purpose    : This method provides the ability to report white spaces
    * Parameters : char[] ch, int start, int end
    * throws     : SAXException when things go wrong
    */

   public void ignorableWhitespace(char[] ch, int start, int end)
      throws SAXException {

      String s = new String(ch, start, end);
      System.out.println("ignorableWhitespace: [" + s + "]");
   }

   /**
    * Method     : characters
    *  Purpose     : This method will report an entity that is skipped by the
parser.
    *               This should only occur for non-validating parsers, and then
is still
    *               implementation-dependent behavior.
    * Parameters : String name
    * throws     : SAXException when things go wrong
    */

   public void skippedEntity(String name) throws SAXException {
      System.out.println("Skipping entity " + name);
   }

} // end of SAXParserDemo
```

```
/**
 * Class  : MyErrorHandler
 * Purpose: implements the SAX ErrorHandler interface.
 *
 */

class MyErrorHandler implements ErrorHandler  {

   /**
    * Method     : warning
    * Purpose     : This method will report a warning that has occurred; this
indicates
    *               that while no XML rules were "broken", something appears
    *               to be incorrect or missing.an entity that is skipped by the
parser.
    * Parameters : SAXParseException exception
    * throws     : SAXException when things go wrong
    */

   public void warning(SAXParseException exception)
      throws SAXException {

      System.out.println("**Parsing Warning**\n" +
                         "  Line:    " +
                            exception.getLineNumber() + "\n" +
                         "  URI:     " +
                            exception.getSystemId() + "\n" +
                         "  Message: " +
                            exception.getMessage());
      throw new SAXException("Warning encountered");
   }

   /**
    * Method     : error
    * Purpose     : This method will report an error if a non-critical parsing
error
    *               has occurred. This error indicates that even if an XML
rules was
    *               broken, the parsing can  continue.
    * Parameters : SAXParseException exception
    * throws     : SAXException when things go wrong
    */

   public void error(SAXParseException exception)
      throws SAXException {

      System.out.println("**Parsing Error**\n" +
                         "  Line:    " +
                            exception.getLineNumber() + "\n" +
                         "  URI:     " +
                            exception.getSystemId() + "\n" +
                         "  Message: " +
                            exception.getMessage());
      throw new SAXException("Error encountered");
   }

   /**
    * Method     : fatalError
    * Purpose    : This method will report a fatal error if a critical parsing
    *              error has occurred. This fatal error indicates that the
    *              parsing process can't be continued because of a major
violation
    *               to XML rules.
    * Parameters : SAXParseException exception
    * throws     : SAXException when things go wrong
    */

   public void fatalError(SAXParseException exception)
```

```
      throws SAXException {

      System.out.println("**Parsing Fatal Error**\n" +
                         "   Line:    " +
                            exception.getLineNumber() + "\n" +
                         "   URI:     " +
                            exception.getSystemId() + "\n" +
                         "   Message: " +
                            exception.getMessage());
      throw new SAXException("Fatal Error encountered");
   }  // end of method fatalError

} // end of class MyErrorHandler


// end of SAXParserDemo.java
```

APPENDIX B. THE APACHE XML SAX PARSER SOURCE CODE

```
/*
 * The Apache Software License, Version 1.1
 *
 *
 * Copyright (c) 1999,2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Xerces" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation and was
 * originally based on software copyright (c) 1999, International
 * Business Machines, Inc., http://www.apache.org.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 */

package org.apache.xerces.parsers;

import org.apache.xerces.framework.XMLAttrList;
import org.apache.xerces.framework.XMLContentSpec;
import org.apache.xerces.framework.XMLDocumentHandler;
import org.apache.xerces.framework.XMLParser;
import org.apache.xerces.readers.XMLEntityHandler;
import org.apache.xerces.utils.QName;
import org.apache.xerces.utils.StringPool;
import org.apache.xerces.validators.common.XMLAttributeDecl;
import org.apache.xerces.validators.common.XMLElementDecl;
```

```
import org.xml.sax.Attributes;
import org.xml.sax.AttributeList;
import org.xml.sax.ContentHandler;
import org.xml.sax.DocumentHandler;
import org.xml.sax.DTDHandler;
import org.xml.sax.EntityResolver;
import org.xml.sax.Parser;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.SAXNotRecognizedException;
import org.xml.sax.SAXNotSupportedException;
import org.xml.sax.ext.DeclHandler;
import org.xml.sax.ext.LexicalHandler;
import org.xml.sax.helpers.AttributesImpl;

// REVISIT: [SAX2beta] ContentHandler#skippedEntity(String)

/**
 * SAXParser provides a parser which implements the SAX1 and SAX2
 * parser APIs.
 *
 * @version $Id: SAXParser.java,v 1.16 2000/06/28 21:00:10 andyc Exp $
 */
public class SAXParser
    extends XMLParser
    implements XMLDocumentHandler, XMLDocumentHandler.DTDHandler,
               Parser, XMLReader {

    //
    // Constants
    //

    // private

    /** Features recognized by this parser. */
    private static final String RECOGNIZED_FEATURES[] = {
        // SAX2 core
        /*"http://xml.org/sax/features/normalize-text",*/
        /*"http://xml.org/sax/features/use-locator",*/
        "http://xml.org/sax/features/namespace-prefixes",
        "http://xml.org/sax/features/string-interning",
        // Xerces
    };

    /** Properties recognized by this parser. */
    private static final String RECOGNIZED_PROPERTIES[] = {
        // SAX2 core
        "http://xml.org/sax/properties/lexical-handler",
        "http://xml.org/sax/properties/declaration-handler",
        "http://xml.org/sax/properties/dom-node",
        // Xerces
    };

    // debugging

    /** Set to true and recompile to debug callbacks. */
    private static final boolean DEBUG_CALLBACKS = false;

    //
    // Data
    //

    // parser handlers

    /** Document handler. */
    private DocumentHandler fDocumentHandler;

    // parser/xmlreader handlers

    /** DTD handler. */
```

```
    private org.xml.sax.DTDHandler fDTDHandler;

    // xmlreader handlers

    /** Content handler. */
    private ContentHandler fContentHandler;

    /** Decl handler. */
    private DeclHandler fDeclHandler;

    /** Lexical handler. */
    private LexicalHandler fLexicalHandler;

    private boolean fNamespacePrefixes = false;

    // temp

    private transient AttributesImpl fAttributes = new AttributesImpl();

    //
    // Constructors
    //

    /** Default constructor. */
    public SAXParser() {
        initHandlers(true, this, this);
    }

    //
    // Public methods
    //

    // features and properties

    /**
     * Returns a list of features that this parser recognizes.
     * This method will never return null; if no features are
     * recognized, this method will return a zero length array.
     *
     * @see #isFeatureRecognized
     * @see #setFeature
     * @see #getFeature
     */
    public String[] getFeaturesRecognized() {

        // get features that super/this recognizes
        String superRecognized[] = super.getFeaturesRecognized();
        String thisRecognized[] = RECOGNIZED_FEATURES;

        // is one or the other the empty set?
        int thisLength = thisRecognized.length;
        if (thisLength == 0) {
            return superRecognized;
        }
        int superLength = superRecognized.length;
        if (superLength == 0) {
            return thisRecognized;
        }

        // combine the two lists and return
        String recognized[] = new String[superLength + thisLength];
        System.arraycopy(superRecognized, 0, recognized, 0, superLength);
        System.arraycopy(thisRecognized,   0,   recognized,   superLength,
thisLength);
        return recognized;

    } // getFeaturesRecognized():String[]

    /**
     * Returns a list of properties that this parser recognizes.
```

```
     * This method will never return null; if no properties are
     * recognized, this method will return a zero length array.
     *
     * @see #isPropertyRecognized
     * @see #setProperty
     * @see #getProperty
     */
    public String[] getPropertiesRecognized() {

        // get properties that super/this recognizes
        String superRecognized[] = super.getPropertiesRecognized();
        String thisRecognized[] = RECOGNIZED_PROPERTIES;

        // is one or the other the empty set?
        int thisLength = thisRecognized.length;
        if (thisLength == 0) {
            return superRecognized;
        }
        int superLength = superRecognized.length;
        if (superLength == 0) {
            return thisRecognized;
        }

        // combine the two lists and return
        String recognized[] = new String[superLength + thisLength];
        System.arraycopy(superRecognized, 0, recognized, 0, superLength);
        System.arraycopy(thisRecognized,   0,   recognized,   superLength,
thisLength);
        return recognized;

    }

    //
    // Protected methods
    //

    // SAX2 core features

    /**
     * <b>Note: Currently, the parser does not support this feature.</b>
     *    Setting    this    feature    to    true    will    throw    a
SAXNotSupportedException.
     * <p>
     * Ensures that all consecutive text is returned in a single callback
     *        to        the        DocumentHandler.characters        or
DocumentHandler.ignorableWhitespace
     * methods.
     * <p>
     * This method is the equivalent to the feature:
     * <pre>
     * http://xml.org/sax/features/normalize-text
     * <pre>
     *
     * @param normalize True to normalize; false not to normalize.
     *
     * @see #getNormalizeText
     * @see #setFeature
     */
    /*
    protected void setNormalizeText(boolean normalize)
        throws SAXNotRecognizedException, SAXNotSupportedException {
        if (normalize) {
            throw                                                    new
SAXNotSupportedException("http://xml.org/sax/features/normalize-text");
        }
    }
    */

    /**
     * <b>Note: This feature is always false.</b>
```

```
             * <p>
             * Returns true if the parser normalizes all consecutive text into
             * a single callback to the DocumentHandler.characters or
             * DocumentHandler.ignorableWhitespace methods.
             *
             * @see #setNormalizeText
             */
            /*
            protected boolean getNormalizeText()
                throws SAXNotRecognizedException, SAXNotSupportedException {
                return false;
            }
            */

            /**
             * <b>Note: Currently, this parser always sets the locator.</b>
             *    Setting    this    feature    to    false    will    throw    a
SAXNotSupportedException.
             * <p>
             * Provide a Locator using the DocumentHandler.setDocumentLocator
             * callback (true), or explicitly do not provide one (false).
             * <p>
             * This method is the equivalent to the feature:
             * <pre>
             * http://xml.org/sax/features/use-locator
             * </pre>
             *
             * @see #getUseLocator
             * @see #setFeature
             */
            /*
            protected void setUseLocator(boolean use)
                throws SAXNotRecognizedException, SAXNotSupportedException {
                if (!use) {
                    throw                                                    new
SAXNotSupportedException("http://xml.org/sax/features/use-locator");
                }
            }
            */

            /**
             * <b>Note: This feature is always true.</b>
             * <p>
             * Returns true if the locator is always set.
             *
             * @see #setUseLocator
             */
            /*
            protected boolean getUseLocator()
                throws SAXNotRecognizedException, SAXNotSupportedException {
                return true;
            }
            */

            // SAX2 core properties

            /**
             * Set the DTD declaration event handler.
             * <p>
             * This method is the equivalent to the property:
             * <pre>
             * http://xml.org/sax/properties/declaration-handler
             * </pre>
             *
             * @param handler The new handler.
             *
             * @see #getDeclHandler
             * @see #setProperty
             */
            protected void setDeclHandler(DeclHandler handler)
```

```
            throws SAXNotRecognizedException, SAXNotSupportedException {
            if (fParseInProgress) {
                throw new SAXNotSupportedException(
                    "PAR011                                                    Feature:
http://xml.org/sax/properties/declaration-handler"
                    +" is not supported during parse."
                    +"\nhttp://xml.org/sax/properties/declaration-handler");
            }
            fDeclHandler = handler;
        }

        /**
         * Returns the DTD declaration event handler.
         *
         * @see #setDeclHandler
         */
        protected DeclHandler getDeclHandler()
            throws SAXNotRecognizedException, SAXNotSupportedException {
            return fDeclHandler;
        }

        /**
         * Set the lexical event handler.
         * <p>
         * This method is the equivalent to the property:
         * <pre>
         * http://xml.org/sax/properties/lexical-handler
         * </pre>
         *
         * @param handler lexical event handler
         *
         * @see #getLexicalHandler
         * @see #setProperty
         */
        protected void setLexicalHandler(LexicalHandler handler)
            throws SAXNotRecognizedException, SAXNotSupportedException {
            if (fParseInProgress) {
                throw new SAXNotSupportedException(
                "PAR011    Feature:    http://xml.org/sax/properties/lexical-
handler"
                +" is not supported during parse."
                +"\nhttp://xml.org/sax/properties/lexical-handler");
            }
            fLexicalHandler = handler;
        }

        /**
         * Returns the lexical handler.
         *
         * @see #setLexicalHandler
         */
        protected LexicalHandler getLexicalHandler()
            throws SAXNotRecognizedException, SAXNotSupportedException {
            return fLexicalHandler;
        }

        //
        // Parser methods
        //

        /** Sets the document handler. */
        public void setDocumentHandler(DocumentHandler handler) {
            fDocumentHandler = handler;
        }

        //
        // Parser/XMLReader methods
        //

        /**
```

```
         * Allow an application to register a DTD event handler.
         *
         * <p>If the application does not register a DTD handler, all DTD
         * events reported by the SAX parser will be silently ignored.</p>
         *
         * <p>Applications may register a new or different handler in the
         * middle of a parse, and the SAX parser must begin using the new
         * handler immediately.</p>
         *
         * @param handler The DTD handler.
         * @exception java.lang.NullPointerException If the handler
         *            argument is null.
         * @see #getDTDHandler
         */
        public void setDTDHandler(org.xml.sax.DTDHandler handler) {
            fDTDHandler = handler;
        }

        /**
         * Return the current DTD handler.
         *
         * @return The current DTD handler, or null if none
         *         has been registered.
         * @see #setDTDHandler
         */
        public org.xml.sax.DTDHandler getDTDHandler() {
            return fDTDHandler;
        }

        /**
         * Sets how the parser reports raw prefixed names,
         * and whether xmlns attributes are reported.
         * <p>
         * This method is the equivalent to the feature:
         * <pre>
         * http://xml.org/sax/features/namespaces-prefixes
         * <pre>
         *
         * @param process True to process namespaces; false to not process.
         *
         * @see #getNamespaces
         * @see #setFeature
         */
        protected void setNamespacePrefixes(boolean process)
            throws SAXNotRecognizedException, SAXNotSupportedException {
            if (fParseInProgress) {
                throw     new     SAXNotSupportedException("PAR004     Cannot
setFeature(http://xml.org/sax/features/namespace-prefixes):    parse    is    in
progress.\n"+

"http://xml.org/sax/features/namespace-prefixes");
            }
            fNamespacePrefixes = process;
        }

        /**
         * Returns the http://xml.org/features/namespace-prefixes
         * value.
         *
         * @see #setNamespacePrefixes
         */
        protected boolean getNamespacePrefixes()
            throws SAXNotRecognizedException, SAXNotSupportedException {
            return fNamespacePrefixes;
        }


        //
        // XMLReader methods
        //
```

```
        /**
         * Set the state of any feature in a SAX2 parser.  The parser
         * might not recognize the feature, and if it does recognize
         * it, it might not be able to fulfill the request.
         *
         * @param featureId The unique identifier (URI) of the feature.
         * @param state The requested state of the feature (true or false).
         *
         * @exception SAXNotRecognizedException If the
         *            requested feature is not known.
         * @exception SAXNotSupportedException If the
         *            requested feature is known, but the requested
         *            state is not supported.
         */
        public void setFeature(String featureId, boolean state)
            throws SAXNotRecognizedException, SAXNotSupportedException {

            //
            // SAX2 Features
            //

            if (featureId.startsWith(SAX2_FEATURES_PREFIX)) {
                String                              feature                         =
featureId.substring(SAX2_FEATURES_PREFIX.length());

                /*
                //
                // http://xml.org/sax/features/normalize-text
                //   Ensure that all consecutive text is returned in a single
callback to
                //                          DocumentHandler.characters       or
DocumentHandler.ignorableWhitespace
                //   (true) or explicitly do not require it (false).
                //
                if (feature.equals("normalize-text")) {
                    setNormalizeText(state);
                    return;
                }
                */
                /*
                //
                // http://xml.org/sax/features/use-locator
                //                  Provide     a     Locator     using     the
DocumentHandler.setDocumentLocator
                //     callback  (true),  or  explicitly  do  not  provide  one
(false).
                //
                if (feature.equals("use-locator")) {
                    setUseLocator(state);
                    return;
                }
                */

                // http://xml.org/sax/features/namespace-prefixes
                //     controls  the  reporting  of  raw  prefixed  names  and
Namespace
                //    declarations (xmlns* attributes): when this feature is
false
                //     (the default), raw prefixed names may optionally be
reported,
                //   and xmlns* attributes must not be reported.
                //
                if (feature.equals("namespace-prefixes")) {
                    setNamespacePrefixes(state);
                    return;
                }
                // http://xml.org/sax/features/string-interning
                //     controls  the  use  of  java.lang.String#intern()  for
strings
```

```
                //   passed to SAX handlers.
                //
                if (feature.equals("string-interning")) {
                    if (state) {
                        throw new SAXNotSupportedException(
                            "PAR018    "+state+"    state    for    feature
\""+featureId+"\" is not supported.\n"+
                            state+'\t'+featureId
                            );
                    }
                    return;
                }

                //
                // Drop through and perform default processing
                //
            }

            //
            // Xerces Features
            //

            /*
            else if (featureId.startsWith(XERCES_FEATURES_PREFIX)) {
                String                          feature                         =
featureId.substring(XERCES_FEATURES_PREFIX.length());
                //
                // Drop through and perform default processing
                //
            }
            */

            //
            // Perform default processing
            //

            super.setFeature(featureId, state);

        } // setFeature(String,boolean)

        /**
         * Query the state of a feature.
         *
         * Query the current state of any feature in a SAX2 parser.  The
         * parser might not recognize the feature.
         *
         * @param featureId The unique identifier (URI) of the feature
         *                  being set.
         * @return The current state of the feature.
         * @exception org.xml.sax.SAXNotRecognizedException If the
         *            requested feature is not known.
         * @exception SAXNotSupportedException If the
         *            requested feature is known but not supported.
         */
        public boolean getFeature(String featureId)
            throws SAXNotRecognizedException, SAXNotSupportedException {

            //
            // SAX2 Features
            //

            if (featureId.startsWith(SAX2_FEATURES_PREFIX)) {
                String                          feature                         =
featureId.substring(SAX2_FEATURES_PREFIX.length());

                /*
                //
                // http://xml.org/sax/features/normalize-text
                //   Ensure that all consecutive text is returned in a single
callback to
```

```
                //                          DocumentHandler.characters      or
DocumentHandler.ignorableWhitespace
                //   (true) or explicitly do not require it (false).
                //
                if (feature.equals("normalize-text")) {
                    return getNormalizeText();
                }
                */
                /*
                //
                // http://xml.org/sax/features/use-locator
                //                Provide     a     Locator     using     the
DocumentHandler.setDocumentLocator
                //     callback  (true),  or  explicitly  do  not  provide  one
(false).
                //
                if (feature.equals("use-locator")) {
                    return getUseLocator();
                }
                */

                // http://xml.org/sax/features/namespace-prefixes
                //     controls  the  reporting  of  raw  prefixed  names  and
Namespace
                //    declarations (xmlns* attributes): when this feature is
false
                //     (the default),  raw prefixed names may optionally be
reported,
                //   and xmlns* attributes must not be reported.
                //
                if (feature.equals("namespace-prefixes")) {
                    return getNamespacePrefixes();
                }
                // http://xml.org/sax/features/string-interning
                //     controls  the  use  of  java.lang.String#intern()  for
strings
                //   passed to SAX handlers.
                //
                if (feature.equals("string-interning")) {
                    return false;
                }

                //
                // Drop through and perform default processing
                //
            }

            //
            // Xerces Features
            //

            /*
            else if (featureId.startsWith(XERCES_FEATURES_PREFIX)) {
                //
                // Drop through and perform default processing
                //
            }
            */

            //
            // Perform default processing
            //

            return super.getFeature(featureId);

        } // getFeature(String):boolean

        /**
         * Set the value of any property in a SAX2 parser.  The parser
         * might not recognize the property, and if it does recognize
```

```
     * it, it might not support the requested value.
     *
     * @param propertyId The unique identifier (URI) of the property
     *                   being set.
     * @param Object The value to which the property is being set.
     *
     * @exception SAXNotRecognizedException If the
     *            requested property is not known.
     * @exception SAXNotSupportedException If the
     *            requested property is known, but the requested
     *            value is not supported.
     */
    public void setProperty(String propertyId, Object value)
        throws SAXNotRecognizedException, SAXNotSupportedException {

        //
        // SAX2 core properties
        //

        if (propertyId.startsWith(SAX2_PROPERTIES_PREFIX)) {
            String                          property                        =
propertyId.substring(SAX2_PROPERTIES_PREFIX.length());
            //
            // http://xml.org/sax/properties/lexical-handler
            // Value type: org.xml.sax.ext.LexicalHandler
            // Access: read/write, pre-parse only
            //   Set the lexical event handler.
            //
            if (property.equals("lexical-handler")) {
                try {
                    setLexicalHandler((LexicalHandler)value);
                }
                catch (ClassCastException e) {
                    throw new SAXNotSupportedException(
                    "PAR012 For propertyID \""
                    +propertyId+"\", the value \""
                    +value+"\" cannot be cast to LexicalHandler."
                    +'\n'+propertyId+'\t'+value+"\tLexicalHandler");
                }
                return;
            }
            //
            // http://xml.org/sax/properties/declaration-handler
            // Value type: org.xml.sax.ext.DeclHandler
            // Access: read/write, pre-parse only
            //   Set the DTD declaration event handler.
            //
            if (property.equals("declaration-handler")) {
                try {
                    setDeclHandler((DeclHandler)value);
                }
                catch (ClassCastException e) {
                    throw new SAXNotSupportedException(
                    "PAR012 For propertyID \""
                    +propertyId+"\", the value \""
                    +value+"\" cannot be cast to DeclHandler."
                    +'\n'+propertyId+'\t'+value+"\tDeclHandler"
                    );
                }
                return;
            }
            //
            // http://xml.org/sax/properties/dom-node
            // Value type: DOM Node
            // Access: read-only
            //    Get the DOM node currently being visited, if the SAX
parser is
            //   iterating over a DOM tree.  If the parser recognises and
supports
```

```
                //   this property but is not currently visiting a DOM node,
it should
                //      return  null  (this  is  a  good  way  to  check  for
availability before the
                //   parse begins).
                //
                if (property.equals("dom-node")) {
                    throw new SAXNotSupportedException(
                        "PAR013 Property \""+propertyId+"\" is read only."
                        +'\n'+propertyId
                        ); // read-only property
                }
                //
                // Drop through and perform default processing
                //
            }

            //
            // Xerces Properties
            //

            /*
            else if (propertyId.startsWith(XERCES_PROPERTIES_PREFIX)) {
                //
                // Drop through and perform default processing
                //
            }
            */

            //
            // Perform default processing
            //

            super.setProperty(propertyId, value);

        } // setProperty(String,Object)

        /**
         * Query the value of a property.
         *
         * Return the current value of a property in a SAX2 parser.
         * The parser might not recognize the property.
         *
         * @param propertyId The unique identifier (URI) of the property
         *                   being set.
         * @return The current value of the property.
         * @exception org.xml.sax.SAXNotRecognizedException If the
         *            requested property is not known.
         * @exception SAXNotSupportedException If the
         *            requested property is known but not supported.
         */
        public Object getProperty(String propertyId)
            throws SAXNotRecognizedException, SAXNotSupportedException {

            //
            // SAX2 core properties
            //

            if (propertyId.startsWith(SAX2_PROPERTIES_PREFIX)) {
                String                          property                        =
propertyId.substring(SAX2_PROPERTIES_PREFIX.length());
                //
                // http://xml.org/sax/properties/lexical-handler
                // Value type: org.xml.sax.ext.LexicalHandler
                // Access: read/write, pre-parse only
                //   Set the lexical event handler.
                //
                if (property.equals("lexical-handler")) {
                    return getLexicalHandler();
                }
```

```
                //
                // http://xml.org/sax/properties/declaration-handler
                // Value type: org.xml.sax.ext.DeclHandler
                // Access: read/write, pre-parse only
                //   Set the DTD declaration event handler.
                //
                if (property.equals("declaration-handler")) {
                    return getDeclHandler();
                }
                //
                // http://xml.org/sax/properties/dom-node
                // Value type: DOM Node
                // Access: read-only
                //    Get the DOM node currently being visited, if the SAX
parser is
                //   iterating over a DOM tree.  If the parser recognises and
supports
                //    this property but is not currently visiting a DOM node,
it should
                //      return null (this is a good way to check for
availability before the
                //   parse begins).
                //
                if (property.equals("dom-node")) {
                    throw new SAXNotSupportedException(
                    "PAR014 Cannot getProperty(\""+propertyId
                    +"\". No DOM Tree exists.\n"+propertyId
                    ); // we are not iterating a DOM tree
                }
                //
                // Drop through and perform default processing
                //
            }

            //
            // Xerces properties
            //

            /*
            else if (propertyId.startsWith(XERCES_PROPERTIES_PREFIX)) {
                //
                // Drop through and perform default processing
                //
            }
            */

            //
            // Perform default processing
            //

            return super.getProperty(propertyId);

        } // getProperty(String):Object

        /**
         * Allow an application to register a content event handler.
         *
         * <p>If the application does not register a content handler, all
         * content events reported by the SAX parser will be silently
         * ignored.</p>
         *
         * <p>Applications may register a new or different handler in the
         * middle of a parse, and the SAX parser must begin using the new
         * handler immediately.</p>
         *
         * @param handler The content handler.
         * @exception java.lang.NullPointerException If the handler
         *            argument is null.
         * @see #getContentHandler
         */
```

```
    public void setContentHandler(ContentHandler handler) {
        if (handler == null) {
            throw new NullPointerException();
        }
        fContentHandler = handler;
    }

    /**
     * Return the current content handler.
     *
     * @return The current content handler, or null if none
     *         has been registered.
     * @see #setContentHandler
     */
    public ContentHandler getContentHandler() {
        return fContentHandler;
    }

    //
    // XMLParser methods
    //

    /**
     *  This function will be called when a &lt;!DOCTYPE...&gt;
declaration is
     * encountered.
     */
    public void startDTD(QName rootElement, int publicId, int systemId)
throws Exception {
        if (fLexicalHandler != null || DEBUG_CALLBACKS) {

            // strings
            String name = fStringPool.toString(rootElement.rawname);
            String pubid = fStringPool.toString(publicId);
            String sysid = fStringPool.toString(systemId);

            // perform callback
            if (DEBUG_CALLBACKS) {
                System.err.println("startDTD(" + name + ", " + pubid + ",
" + sysid + ")");
            }
            if (fLexicalHandler != null) {
                fLexicalHandler.startDTD(name, pubid, sysid);
            }
        }
    }

    /**
     *  This function will be called at the end of the DTD.
     */
    public void endDTD() throws Exception {
        if (DEBUG_CALLBACKS) {
            System.err.println("endDTD()");
        }
        if (fLexicalHandler != null) {
            fLexicalHandler.endDTD();
        }
    }

    /**
     * Report an element type declaration.
     *
     * The content model will consist of the string "EMPTY", the
     * string "ANY", or a parenthesised group, optionally followed
     * by an occurrence indicator.  The model will be normalized so
     * that all whitespace is removed.
     *
     * @param name The element type name.
     * @param model The content model as a normalized string.
     * @exception SAXException The application may raise an exception.
```

```
         */
        public void elementDecl(QName elementDecl,
                                int contentSpecType,
                                int contentSpecIndex,
                                XMLContentSpec.Provider  contentSpecProvider)
throws Exception {

            if (fDeclHandler != null || DEBUG_CALLBACKS) {

                // strings
                String name = fStringPool.toString(elementDecl.rawname);
                String contentModel;
                if (contentSpecType == XMLElementDecl.TYPE_ANY) {
                    contentModel = "ANY";
                }
                else if (contentSpecType == XMLElementDecl.TYPE_EMPTY) {
                    contentModel = "EMPTY";
                }
                else {
                    contentModel                                              =
XMLContentSpec.toString(contentSpecProvider,
                                                          fStringPool,
                                                          contentSpecIndex);
                }

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("elementDecl("  +  name  +  ",  "  +
contentModel + ")");
                }
                if (fDeclHandler != null) {
                    fDeclHandler.elementDecl(name, contentModel);
                }
            }

        }

        /**
         * Report an attribute type declaration.
         *
         * Only the effective (first) declaration for an attribute will
         * be reported.  The type will be one of the strings "CDATA",
         * "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY",
         * "ENTITIES", or "NOTATION", or a parenthesized token group with
         * the separator "|" and all whitespace removed.
         *
         * @param eName The name of the associated element.
         * @param aName The name of the attribute.
         * @param type A string representing the attribute type.
         * @param valueDefault A string representing the attribute default
         *        ("#IMPLIED", "#REQUIRED", or "#FIXED") or null if
         *        none of these applies.
         * @param value A string representing the attribute's default value,
         *        or null if there is none.
         * @exception SAXException The application may raise an exception.
         */
        public void attlistDecl(QName elementDecl, QName attributeDecl,
                                int   attType,   boolean   attList,   String
enumString,
                                int attDefaultType,
                                int attDefaultValue) throws Exception
        {
            if (fDeclHandler != null || DEBUG_CALLBACKS) {

                // strings
                String eName = fStringPool.toString(elementDecl.rawname);
                String aName = fStringPool.toString(attributeDecl.rawname);
                String aType = enumString;
                if (attType != XMLAttributeDecl.TYPE_ENUMERATION) {
                    switch (attType) {
```

```
                        case XMLAttributeDecl.TYPE_CDATA: {
                            aType = "CDATA";
                            break;
                        }
                        case XMLAttributeDecl.TYPE_ENTITY: {
                            aType = attList ? "ENTITIES" : "ENTITY";
                            break;
                        }
                        case XMLAttributeDecl.TYPE_ID: {
                            aType = "ID";
                            break;
                        }
                        case XMLAttributeDecl.TYPE_IDREF: {
                            aType = attList ? "IDREFS" : "IDREF";
                            break;
                        }
                        case XMLAttributeDecl.TYPE_NMTOKEN: {
                            aType = attList ? "NMTOKENS" : "NMTOKEN";
                            break;
                        }
                        case XMLAttributeDecl.TYPE_NOTATION: {
                            aType = "NOTATION";
                            break;
                        }
                    }
                }
                String aDefaultType = "";
                switch (attDefaultType) {
                    case XMLAttributeDecl.DEFAULT_TYPE_FIXED: {
                        aDefaultType = "#FIXED";
                        break;
                    }
                    case XMLAttributeDecl.DEFAULT_TYPE_IMPLIED: {
                        aDefaultType = "#IMPLIED";
                        break;
                    }
                    case XMLAttributeDecl.DEFAULT_TYPE_REQUIRED: {
                        aDefaultType = "#REQUIRED";
                        break;
                    }
                }
                String aDefaultValue = fStringPool.toString(attDefaultValue);

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("attributeDecl(" +
                                       eName + ", " +
                                       aName + ", " +
                                       aType + ", " +
                                       aDefaultType + ", " +
                                       aDefaultValue + ")");
                }
                if (fDeclHandler != null) {
                    fDeclHandler.attributeDecl(eName,      aName,      aType,
aDefaultType, aDefaultValue);
                }
            }
        }

        /**
         * Report an internal parameter entity declaration.
         */
        public void internalPEDecl(int entityName, int entityValue) throws
Exception {

            if (fDeclHandler != null || DEBUG_CALLBACKS) {

                // strings
                String name = "%" + fStringPool.toString(entityName);
                String value = fStringPool.toString(entityValue);
```

```
                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("internalEntityDecl(" + name + ", " +
value + ")");
                    }
                    if (fDeclHandler != null) {
                        fDeclHandler.internalEntityDecl(name, value);
                    }
                }

            }

            /**
             * Report a parsed external parameter entity declaration.
             */
            public void externalPEDecl(int entityName, int publicId,
systemId) throws Exception {

                if (fDeclHandler != null || DEBUG_CALLBACKS) {

                    // strings
                    String name = "%" + fStringPool.toString(entityName);
                    String pubid = fStringPool.toString(publicId);
                    String sysid = fStringPool.toString(systemId);

                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("externalEntityDecl(" + name + ", " +
pubid + ", " + sysid + ")");
                    }
                    if (fDeclHandler != null) {
                        fDeclHandler.externalEntityDecl(name, pubid, sysid);
                    }
                }

            }

            /**
             * Report an internal general entity declaration.
             */
            public void internalEntityDecl(int entityName, int entityValue)
throws Exception {

                if (fDeclHandler != null || DEBUG_CALLBACKS) {

                    // strings
                    String name = fStringPool.toString(entityName);
                    String value = fStringPool.toString(entityValue);

                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("internalEntityDecl(" + name + ", " +
value + ")");
                    }
                    if (fDeclHandler != null) {
                        fDeclHandler.internalEntityDecl(name, value);
                    }
                }

            }

            /**
             * Report a parsed external general entity declaration.
             */
            public void externalEntityDecl(int entityName, int publicId, int
systemId) throws Exception {

                if (fDeclHandler != null || DEBUG_CALLBACKS) {
```

```
                    // strings
                    String name = fStringPool.toString(entityName);
                    String pubid = fStringPool.toString(publicId);
                    String sysid = fStringPool.toString(systemId);

                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("externalEntityDecl(" + name + ", " +
pubid + ", " + sysid + ")");
                    }
                    if (fDeclHandler != null) {
                        fDeclHandler.externalEntityDecl(name, pubid, sysid);
                    }
                }

            }

            /**
             * Receive notification of an unparsed entity declaration event.
             */
            public void unparsedEntityDecl(int entityName, int publicId, int
systemId, int notationName) throws Exception {

                if (fDTDHandler != null || DEBUG_CALLBACKS) {

                    // strings
                    String name = fStringPool.toString(entityName);
                    String pubid = fStringPool.toString(publicId);
                    String sysid = fStringPool.toString(systemId);
                    String notation = fStringPool.toString(notationName);

                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("unparsedEntityDecl(" + name + ", " +
pubid + ", " + sysid + ", " + notation + ")");
                    }
                    if (fDTDHandler != null) {
                        fDTDHandler.unparsedEntityDecl(name,    pubid,    sysid,
notation);
                    }
                }

            }

            /**
             * Receive notification of a notation declaration event.
             */
            public  void  notationDecl(int  notationName,  int  publicId,  int
systemId) throws Exception {

                if (fDTDHandler != null || DEBUG_CALLBACKS) {

                    // strings
                    String name = fStringPool.toString(notationName);
                    String pubid = fStringPool.toString(publicId);
                    String sysid = fStringPool.toString(systemId);

                    // perform callback
                    if (DEBUG_CALLBACKS) {
                        System.err.println("notationDecl(" + name + ", " + pubid
+ ", " + sysid + ")");
                    }
                    if (fDTDHandler != null) {
                        fDTDHandler.notationDecl(name, pubid, sysid);
                    }
                }

            }

            /** Start document. */
```

```
        public void startDocument() throws Exception {

            // perform callbacks
            if (DEBUG_CALLBACKS) {
                System.err.println("setDocumentLocator(<locator>)");
                System.err.println("startDocument()");
            }
            if (fDocumentHandler != null) {
                fDocumentHandler.setDocumentLocator(getLocator());
                fDocumentHandler.startDocument();
            }
            if (fContentHandler != null) {
                fContentHandler.setDocumentLocator(getLocator());
                fContentHandler.startDocument();
            }

        } // startDocument()

        /** End document. */
        public void endDocument() throws Exception {

            // perform callback
            if (DEBUG_CALLBACKS) {
                System.err.println("endDocument()");
            }
            if (fDocumentHandler != null) {
                fDocumentHandler.endDocument();
            }
            if (fContentHandler != null) {
                fContentHandler.endDocument();
            }

        } // endDocument()

        /** XML declaration. */
        public void xmlDecl(int versionIndex, int encodingIndex, int
standaloneIndex) throws Exception {

            // perform callbacks
            if (DEBUG_CALLBACKS) {
                String notes = "";
                if (versionIndex != -1)
                    notes += " version='"
+
fStringPool.toString(versionIndex) + "'";
                if (encodingIndex != -1)
                    notes += " encoding='"
+
fStringPool.toString(encodingIndex) + "'";
                if (standaloneIndex != -1)
                    notes += " standalone='"
+
fStringPool.toString(standaloneIndex) + "'";
                System.err.println("xmlDecl(<?xml" + notes + "?>)");
            }

            // release strings
            fStringPool.releaseString(versionIndex);
            fStringPool.releaseString(encodingIndex);
            fStringPool.releaseString(standaloneIndex);

        }

        /** Text declaration. */
        public void textDecl(int versionIndex, int encodingIndex) throws
Exception {

            // perform callbacks
            if (DEBUG_CALLBACKS) {
                String notes = "";
                if (versionIndex != -1)
                    notes += " version='"
+
fStringPool.toString(versionIndex) + "'";
```

```
                if (encodingIndex != -1)
                    notes          +=          "          encoding='"          +
fStringPool.toString(encodingIndex) + "'";
                System.err.println("textDecl(<?xml" + notes + "?>)");
            }

            // release strings
            fStringPool.releaseString(versionIndex);
            fStringPool.releaseString(encodingIndex);
        }

        /**
         * Report the start of the scope of a namespace declaration.
         */
        public  void  startNamespaceDeclScope(int  prefix,  int  uri)  throws
Exception {

            if (fContentHandler != null || DEBUG_CALLBACKS) {

                // strings
                String p = fStringPool.toString(prefix);
                String ns = fStringPool.toString(uri);

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("startNamespaceDeclScope(" + p + ", "
+ ns + ")");
                }
                if (fContentHandler != null) {
                    fContentHandler.startPrefixMapping(p, ns);
                }
            }

        }

        /**
         * Report the end of the scope of a namespace declaration.
         */
        public void endNamespaceDeclScope(int prefix) throws Exception {

            if (fContentHandler != null || DEBUG_CALLBACKS) {

                // strings
                String p = fStringPool.toString(prefix);

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("endNamespaceDeclScope(" + p + ")");
                }
                if (fContentHandler != null) {
                    fContentHandler.endPrefixMapping(p);
                }
            }

        }

        /** New callback from DOM Level 2. There is no corresponding SAX
callout for this yet. */
        public void internalSubset(int internalSubset) {
        }

        /** Start element */
        public void startElement(QName element,
                                 XMLAttrList attrList, int attrListIndex)
            throws Exception {

            // parameters
            String name = fStringPool.toString(element.rawname);
            AttributeList attrs = attrList.getAttributeList(attrListIndex);
```

```
                // perform callback
                if (DEBUG_CALLBACKS) {
                    String atts = attrs.getLength() > 0 ? "" : " ";
                    for (int i = 0; i < attrs.getLength(); i++) {
                        atts += " " + attrs.getName(i) + "='" + attrs.getValue(i
+ "'";
                    }
                    System.err.println("startElement(" + name + "," + atts +
")");
                }
                if (fDocumentHandler != null) {
                    fDocumentHandler.startElement(name, attrs);
                }
                if (fContentHandler != null) {
                    boolean namespaces = getNamespaces();
                    int uriIndex = element.uri;
                    String uri = uriIndex != -1 && namespaces
                               ? fStringPool.toString(uriIndex) : "";
                    int localIndex = element.localpart;
                    String local = localIndex != -1 && namespaces
                                 ? fStringPool.toString(localIndex) : "";
                    String raw = name;
                    fAttributes.clear();
                    for (int attrIndex = attrList.getFirstAttr(attrListIndex);
                         attrIndex != -1;
                         attrIndex = attrList.getNextAttr(attrIndex)) {
                        int attrNameIndex = attrList.getAttrName(attrIndex);
                        int attrUriIndex = attrList.getAttrURI(attrIndex);
                        String attrUri = attrUriIndex != -1 && namespaces
                                       ? fStringPool.toString(attrUriIndex) : "";
                        int                    attrLocalIndex                  =
attrList.getAttrLocalpart(attrIndex);
                        String attrLocal = attrLocalIndex != -1 && namespaces
                                         ? fStringPool.toString(attrLocalIndex) :
"";
                        String attrRaw = fStringPool.toString(attrNameIndex);
                        String                      attrType                    =
fStringPool.toString(attrList.getAttType(attrIndex));
                        String                     attrValue                    =
fStringPool.toString(attrList.getAttValue(attrIndex));
                        //int                     attrPrefix                    =
fStringPool.getPrefixForQName(attrNameIndex);
                        int attrPrefix = attrList.getAttrPrefix(attrIndex);
                        boolean namespacePrefixes = getNamespacePrefixes();
                        if (!namespaces || namespacePrefixes ||
                            (attrPrefix != fStringPool.addSymbol("xmlns")
                            && attrLocalIndex != fStringPool.addSymbol("xmlns")
                            ))
                            fAttributes.addAttribute(attrUri, attrLocal, attrRaw,
                                                     attrType, attrValue);

                    }
                    fContentHandler.startElement(uri, local, raw, fAttributes);
                }

                // free attribute list
                attrList.releaseAttrList(attrListIndex);

            } // startElement(QName,XMLAttrList,int)

            /** End element. */
            public void endElement(QName element) throws Exception {

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("endElement("                          +
fStringPool.toString(element.rawname) + ")");
                }
                if (fDocumentHandler != null) {
```

```
fDocumentHandler.endElement(fStringPool.toString(element.rawname));
                }
                if (fContentHandler != null) {
                    boolean namespaces = getNamespaces();
                    int uriIndex = element.uri;
                    String uri = uriIndex != -1 && namespaces
                               ? fStringPool.toString(uriIndex) : "";
                    int localIndex = element.localpart;
                    String local = localIndex != -1 && namespaces
                                 ? fStringPool.toString(localIndex) : "";
                    String raw = fStringPool.toString(element.rawname);
                    fContentHandler.endElement(uri, local, raw);
                }

            } // endElement(QName)

            /** Start entity reference. */
            public void startEntityReference(int entityName, int entityType, int
entityContext) throws Exception {
                if (fLexicalHandler != null || DEBUG_CALLBACKS) {
                    switch (entityType) {
                    case XMLEntityHandler.ENTITYTYPE_INTERNAL_PE:
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL_PE:
                        if (DEBUG_CALLBACKS) {
                            System.err.println("startEntity(%"                    +
fStringPool.toString(entityName) + ")");
                        }
                        if (fLexicalHandler != null) {
                            fLexicalHandler.startEntity("%"                      +
fStringPool.toString(entityName));
                        }
                        break;
                    case XMLEntityHandler.ENTITYTYPE_INTERNAL:
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL:
                        if (DEBUG_CALLBACKS) {
                            System.err.println("startEntity("                     +
fStringPool.toString(entityName) + ")");
                        }
                        if (fLexicalHandler != null) {

fLexicalHandler.startEntity(fStringPool.toString(entityName));
                        }
                        break;
                    case  XMLEntityHandler.ENTITYTYPE_UNPARSED:     //  these  are
mentioned by name, not referenced
                        throw new RuntimeException(
                            "PAR015                          startEntityReference():
ENTITYTYPE_UNPARSED");
                    case XMLEntityHandler.ENTITYTYPE_DOCUMENT:
                        break;                    // not reported
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL_SUBSET:
                        if (DEBUG_CALLBACKS) {
                            System.err.println("startEntity(\"[dtd]\")");
                        }
                        if (fLexicalHandler != null) {
                            fLexicalHandler.startEntity("[dtd]");
                        }
                        break;
                    }
                }
            }

            /** End entity reference. */
            public void endEntityReference(int entityName, int entityType, int
entityContext) throws Exception {
                if (fLexicalHandler != null || DEBUG_CALLBACKS) {
                    switch (entityType) {
                    case XMLEntityHandler.ENTITYTYPE_INTERNAL_PE:
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL_PE:
```

```
                        if (DEBUG_CALLBACKS) {
                            System.err.println("endEntity(%"                    +
fStringPool.toString(entityName) + ")");
                        }
                        if (fLexicalHandler != null) {
                            fLexicalHandler.endEntity("%"                      +
fStringPool.toString(entityName));
                        }
                        break;
                    case XMLEntityHandler.ENTITYTYPE_INTERNAL:
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL:
                        if (DEBUG_CALLBACKS) {
                            System.err.println("endEntity("                    +
fStringPool.toString(entityName) + ")");
                        }
                        if (fLexicalHandler != null) {

fLexicalHandler.endEntity(fStringPool.toString(entityName));
                        }
                        break;
                    case  XMLEntityHandler.ENTITYTYPE_UNPARSED:     //  these  are
mentioned by name, not referenced
                        throw new RuntimeException("PAR016 endEntityReference():
ENTITYTYPE_UNPARSED");
                    case XMLEntityHandler.ENTITYTYPE_DOCUMENT:
                        break;                  // not reported
                    case XMLEntityHandler.ENTITYTYPE_EXTERNAL_SUBSET:
                        if (DEBUG_CALLBACKS) {
                            System.err.println("endEntity(\"[dtd]\")");
                        }
                        if (fLexicalHandler != null) {
                            fLexicalHandler.endEntity("[dtd]");
                        }
                        break;
                    }
                }
            }

            /** Start CDATA section. */
            public void startCDATA() throws Exception {
                if (DEBUG_CALLBACKS) {
                    System.err.println("startCDATA()");
                }
                if (fLexicalHandler != null) {
                    fLexicalHandler.startCDATA();
                }
            }

            /** End CDATA section. */
            public void endCDATA() throws Exception {
                if (DEBUG_CALLBACKS) {
                    System.err.println("endCDATA()");
                }
                if (fLexicalHandler != null) {
                    fLexicalHandler.endCDATA();
                }
            }

            /** Not called. */
            public void characters(int dataIndex) throws Exception {
                throw new RuntimeException("PAR017 cannot happen 5\n5");
            }

            /** Not called. */
            public void ignorableWhitespace(int dataIndex) throws Exception {
                throw new RuntimeException("PAR017 cannot happen 6\n6");
            }

            /** Processing instruction. */
```

```
        public void processingInstruction(int piTarget, int piData) throws
Exception {

            if (fDocumentHandler != null || fContentHandler != null ||
DEBUG_CALLBACKS) {
                //
                // REVISIT - I keep running into SAX apps that expect
                //   null data to be an empty string, which is contrary
                //   to the comment for this method in the SAX API.
                //

                // strings
                String target = fStringPool.orphanString(piTarget);
                String    data    =    piData    ==    -1    ?    ""    :
fStringPool.orphanString(piData);

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("processingInstruction(" + target + ",
" + data + ")");
                }
                if (fDocumentHandler != null) {
                    fDocumentHandler.processingInstruction(target, data);
                }
                if (fContentHandler != null) {
                    fContentHandler.processingInstruction(target, data);
                }

            }
            else {
                fStringPool.releaseString(piTarget);
                fStringPool.releaseString(piData);
            }

        }

        /** Comment. */
        public void comment(int dataIndex) throws Exception {

            if (fLexicalHandler != null || DEBUG_CALLBACKS) {

                // strings
                String data = fStringPool.orphanString(dataIndex);

                // perform callback
                if (DEBUG_CALLBACKS) {
                    System.err.println("comment(" + data + ")");
                }
                if (fLexicalHandler != null) {
                    fLexicalHandler.comment(data.toCharArray(),            0,
data.length());
                }
            } else {
                fStringPool.releaseString(dataIndex);
            }
        }

        /** Characters. */
        public void characters(char ch[], int start, int length) throws
Exception {

            // perform callback
            if (DEBUG_CALLBACKS) {
                System.err.println("characters(<char-data>)   length   "   +
length);
            }
            if (fDocumentHandler != null) {
                fDocumentHandler.characters(ch, start, length);
            }
            if (fContentHandler != null) {
```

```
                fContentHandler.characters(ch, start, length);
            }

        }

        /** Ignorable whitespace. */
        public void ignorableWhitespace(char ch[], int start, int length)
throws Exception {

            // perform callback
            if (DEBUG_CALLBACKS) {
                System.err.println("ignorableWhitespace(<white-space>)");
            }
            if (fDocumentHandler != null) {
                fDocumentHandler.ignorableWhitespace(ch, start, length);
            }
            if (fContentHandler != null) {
                fContentHandler.ignorableWhitespace(ch, start, length);
            }

        }

    } // class SAXParser
```

APPENDIX C. THE APACHE XML SAX PARSER – CONTENT HANDLER SOURCE CODE

```
// CONTENTHANLDER.JAVA - HANDLE MAIN DOCUMENT CONTENT.
// Written by David Megginson, sax@megginson.com
// NO WARRANTY!  This class is in the public domain.

// $Id: ContentHandler.java,v 1.5 2000/05/05 17:45:39 david Exp $

package org.xml.sax;


/**
 * Receive notification of the logical content of a document.
 *
 * <blockquote>
 * <em>This module, both source code and documentation, is in the
 * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
 * </blockquote>
 *
 * <p>This is the main interface that most SAX applications
 * implement: if the application needs to be informed of basic parsing
 * events, it implements this interface and registers an instance with
 *        the       SAX       parser       using       the       {@link
org.xml.sax.XMLReader#setContentHandler
 * setContentHandler} method.  The parser uses the instance to report
 * basic document-related events like the start and end of elements
 * and character data.</p>
 *
 * <p>The order of events in this interface is very important, and
 * mirrors the order of information in the document itself.  For
 * example, all of an element's content (character data, processing
 * instructions, and/or subelements) will appear, in order, between
 * the startElement event and the corresponding endElement event.</p>
 *
 * <p>This interface is similar to the now-deprecated SAX 1.0
 * DocumentHandler interface, but it adds support for Namespaces
 * and for reporting skipped entities (in non-validating XML
 * processors).</p>
 *
 * <p>Implementors should note that there is also a Java class
 * {@link java.net.ContentHandler ContentHandler} in the java.net
 * package; that means that it's probably a bad idea to do</p>
 *
 * <blockquote>
 * import java.net.*;
 * import org.xml.sax.*;
 * </blockquote>
 *
 * <p>In fact, "import ...*" is usually a sign of sloppy programming
 * anyway, so the user should consider this a feature rather than a
 * bug.</p>
 *
 * @since SAX 2.0
 * @author David Megginson,
 *         <a href="mailto:sax@megginson.com">sax@megginson.com</a>
 * @version 2.0
 * @see org.xml.sax.XMLReader
 * @see org.xml.sax.DTDHandler
 * @see org.xml.sax.ErrorHandler
 */
public interface ContentHandler
{

    /**
     * Receive an object for locating the origin of SAX document events.
     *
     * <p>SAX parsers are strongly encouraged (though not absolutely
     * required) to supply a locator: if it does so, it must supply
     * the locator to the application by invoking this method before
     * invoking any of the other methods in the ContentHandler
```

```
     * interface.</p>
     *
     * <p>The locator allows the application to determine the end
     * position of any document-related event, even if the parser is
     * not reporting an error.  Typically, the application will
     * use this information for reporting its own errors (such as
     * character content that does not match an application's
     * business rules).  The information returned by the locator
     * is probably not sufficient for use with a search engine.</p>
     *
     * <p>Note that the locator will return correct information only
     * during the invocation of the events in this interface.  The
     * application should not attempt to use it at any other time.</p>
     *
     * @param locator An object that can return the location of
     *                any SAX document event.
     * @see org.xml.sax.Locator
     */
    public void setDocumentLocator (Locator locator);


    /**
     * Receive notification of the beginning of a document.
     *
     * <p>The SAX parser will invoke this method only once, before any
     *    other    methods    in    this    interface    or    in    {@link
org.xml.sax.DTDHandler
     * DTDHandler} (except for {@link #setDocumentLocator
     * setDocumentLocator}).</p>
     *
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see #endDocument
     */
    public void startDocument ()
      throws SAXException;


    /**
     * Receive notification of the end of a document.
     *
     * <p>The SAX parser will invoke this method only once, and it will
     * be the last method invoked during the parse.  The parser shall
     * not invoke this method until it has either abandoned parsing
     * (because of an unrecoverable error) or reached the end of
     * input.</p>
     *
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see #startDocument
     */
    public void endDocument()
      throws SAXException;


    /**
     * Begin the scope of a prefix-URI Namespace mapping.
     *
     * <p>The information from this event is not necessary for
     * normal Namespace processing: the SAX XML reader will
     * automatically replace prefixes for element and attribute
     * names when the <code>http://xml.org/sax/features/namespaces</code>
     * feature is <var>true</var> (the default).</p>
     *
     * <p>There are cases, however, when applications need to
     * use prefixes in character data or in attribute values,
     * where they cannot safely be expanded automatically; the
     * start/endPrefixMapping event supplies the information
     * to the application to expand prefixes in those contexts
     * itself, if necessary.</p>
```

```
     *
     * <p>Note that start/endPrefixMapping events are not
     * guaranteed to be properly nested relative to each-other:
     * all startPrefixMapping events will occur before the
     * corresponding {@link #startElement startElement} event,
     * and all {@link #endPrefixMapping endPrefixMapping}
     * events will occur after the corresponding {@link #endElement
     * endElement} event, but their order is not otherwise
     * guaranteed.</p>
     *
     * <p>There should never be start/endPrefixMapping events for the
     * "xml" prefix, since it is predeclared and immutable.</p>
     *
     * @param prefix The Namespace prefix being declared.
     * @param uri The Namespace URI the prefix is mapped to.
     * @exception org.xml.sax.SAXException The client may throw
     *            an exception during processing.
     * @see #endPrefixMapping
     * @see #startElement
     */
    public void startPrefixMapping (String prefix, String uri)
      throws SAXException;


    /**
     * End the scope of a prefix-URI mapping.
     *
     * <p>See {@link #startPrefixMapping startPrefixMapping} for
     * details.  This event will always occur after the corresponding
     * {@link #endElement endElement} event, but the order of
     * {@link #endPrefixMapping endPrefixMapping} events is not otherwise
     * guaranteed.</p>
     *
     * @param prefix The prefix that was being mapping.
     * @exception org.xml.sax.SAXException The client may throw
     *            an exception during processing.
     * @see #startPrefixMapping
     * @see #endElement
     */
    public void endPrefixMapping (String prefix)
      throws SAXException;


    /**
     * Receive notification of the beginning of an element.
     *
     * <p>The Parser will invoke this method at the beginning of every
     * element in the XML document; there will be a corresponding
     * {@link #endElement endElement} event for every startElement event
     *  (even when the element is empty). All of the element's content
will be
     * reported, in order, before the corresponding endElement
     * event.</p>
     *
     * <p>This event allows up to three name components for each
     * element:</p>
     *
     * <ol>
     * <li>the Namespace URI;</li>
     * <li>the local name; and</li>
     * <li>the qualified (prefixed) name.</li>
     * </ol>
     *
     * <p>Any or all of these may be provided, depending on the
     * values of the <var>http://xml.org/sax/features/namespaces</var>
     * and the <var>http://xml.org/sax/features/namespace-prefixes</var>
     * properties:</p>
     *
     * <ul>
     * <li>the Namespace URI and local name are required when
```

```
     * the namespaces property is <var>true</var> (the default), and are
     * optional when the namespaces property is <var>false</var> (if one
is
     * specified, both must be);</li>
     * <li>the qualified name is required when the namespace-prefixes
property
     * is <var>true</var>, and is optional when the namespace-prefixes
property
     * is <var>false</var> (the default).</li>
     * </ul>
     *
     * <p>Note that the attribute list provided will contain only
     * attributes with explicit values (specified or defaulted):
     * #IMPLIED attributes will be omitted.  The attribute list
     * will contain attributes used for Namespace declarations
     * (xmlns* attributes) only if the
     * <code>http://xml.org/sax/features/namespace-prefixes</code>
     * property is true (it is false by default, and support for a
     * true value is optional).</p>
     *
     * @param uri The Namespace URI, or the empty string if the
     *        element has no Namespace URI or if Namespace
     *        processing is not being performed.
     * @param localName The local name (without prefix), or the
     *        empty string if Namespace processing is not being
     *        performed.
     * @param qName The qualified name (with prefix), or the
     *        empty string if qualified names are not available.
     * @param atts The attributes attached to the element.  If
     *        there are no attributes, it shall be an empty
     *        Attributes object.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see #endElement
     * @see org.xml.sax.Attributes
     */
    public void startElement (String namespaceURI, String localName,
                              String qName, Attributes atts)
      throws SAXException;


    /**
     * Receive notification of the end of an element.
     *
     * <p>The SAX parser will invoke this method at the end of every
     * element in the XML document; there will be a corresponding
     * {@link #startElement startElement} event for every endElement
     * event (even when the element is empty).</p>
     *
     * <p>For information on the names, see startElement.</p>
     *
     * @param uri The Namespace URI, or the empty string if the
     *        element has no Namespace URI or if Namespace
     *        processing is not being performed.
     * @param localName The local name (without prefix), or the
     *        empty string if Namespace processing is not being
     *        performed.
     * @param qName The qualified XML 1.0 name (with prefix), or the
     *        empty string if qualified names are not available.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     */
    public void endElement (String namespaceURI, String localName,
                            String qName)
      throws SAXException;


    /**
     * Receive notification of character data.
     *
```

```
     * <p>The Parser will call this method to report each chunk of
     * character data.  SAX parsers may return all contiguous character
     * data in a single chunk, or they may split it into several
     * chunks; however, all of the characters in any single event
     * must come from the same external entity so that the Locator
     * provides useful information.</p>
     *
     * <p>The application must not attempt to read from the array
     * outside of the specified range.</p>
     *
     * <p>Note that some parsers will report whitespace in element
     * content using the {@link #ignorableWhitespace ignorableWhitespace}
     * method rather than this one (validating parsers <em>must</em>
     * do so).</p>
     *
     * @param ch The characters from the XML document.
     * @param start The start position in the array.
     * @param length The number of characters to read from the array.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see #ignorableWhitespace
     * @see org.xml.sax.Locator
     */
    public void characters (char ch[], int start, int length)
      throws SAXException;


    /**
     * Receive notification of ignorable whitespace in element content.
     *
     * <p>Validating Parsers must use this method to report each chunk
     *  of  whitespace  in  element  content  (see  the  W3C  XML  1.0
recommendation,
     * section 2.10): non-validating parsers may also use this method
     * if they are capable of parsing and using content models.</p>
     *
     * <p>SAX parsers may return all contiguous whitespace in a single
     * chunk, or they may split it into several chunks; however, all of
     * the characters in any single event must come from the same
     * external entity, so that the Locator provides useful
     * information.</p>
     *
     * <p>The application must not attempt to read from the array
     * outside of the specified range.</p>
     *
     * @param ch The characters from the XML document.
     * @param start The start position in the array.
     * @param length The number of characters to read from the array.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see #characters
     */
    public void ignorableWhitespace (char ch[], int start, int length)
      throws SAXException;


    /**
     * Receive notification of a processing instruction.
     *
     * <p>The Parser will invoke this method once for each processing
     * instruction found: note that processing instructions may occur
     * before or after the main document element.</p>
     *
     * <p>A SAX parser must never report an XML declaration (XML 1.0,
     * section 2.8) or a text declaration (XML 1.0, section 4.3.1)
     * using this method.</p>
     *
     * @param target The processing instruction target.
     * @param data The processing instruction data, or null if
     *        none was supplied.  The data does not include any
```

```
             *        whitespace separating it from the target.
             * @exception org.xml.sax.SAXException Any SAX exception, possibly
             *            wrapping another exception.
             */
            public void processingInstruction (String target, String data)
              throws SAXException;


            /**
             * Receive notification of a skipped entity.
             *
             * <p>The Parser will invoke this method once for each entity
             * skipped.  Non-validating processors may skip entities if they
             * have not seen the declarations (because, for example, the
             * entity was declared in an external DTD subset).  All processors
             * may skip external entities, depending on the values of the
             * <code>http://xml.org/sax/features/external-general-entities</code>
             * and the
             *              <code>http://xml.org/sax/features/external-parameter-
entities</code>
             * properties.</p>
             *
             * @param name The name of the skipped entity.  If it is a
             *        parameter entity, the name will begin with '%', and if
             *        it is the external DTD subset, it will be the string
             *        "[dtd]".
             * @exception org.xml.sax.SAXException Any SAX exception, possibly
             *            wrapping another exception.
             */
            public void skippedEntity (String name)
              throws SAXException;
        }
        // end of ContentHandler.java
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. THE APACHE XML SAX PARSER – ERROR HANDLER SOURCE CODE

```
// SAX ERROR HANDLER.
      // No warranty; no copyright -- use this as you will.
      // $Id: ErrorHandler.java,v 1.4 2000/05/05 17:46:27 david Exp $

      package org.xml.sax;


      /**
       * Basic interface for SAX error handlers.
       *
       * <blockquote>
       * <em>This module, both source code and documentation, is in the
       * Public Domain, and comes with <strong>NO WARRANTY</strong>.</em>
       * </blockquote>
       *
       * <p>If a SAX application needs to implement customized error
       * handling, it must implement this interface and then register an
       * instance with the XML reader using the
       * {@link org.xml.sax.XMLReader#setErrorHandler setErrorHandler}
       * method.  The parser will then report all errors and warnings
       * through this interface.</p>
       *
       * <p><strong>WARNING:</strong> If an application does <em>not</em>
       * register an ErrorHandler, XML parsing errors will go unreported
       * and bizarre behaviour may result.</p>
       *
       * <p>For XML processing errors, a SAX driver must use this interface
       * instead of throwing an exception: it is up to the application
       * to decide whether to throw an exception for different types of
       * errors and warnings.  Note, however, that there is no requirement that
       * the parser continue to provide useful information after a call to
       * {@link #fatalError fatalError} (in other words, a SAX driver class
       * could catch an exception and report a fatalError).</p>
       *
       * @since SAX 1.0
       * @author David Megginson,
       *         <a href="mailto:sax@megginson.com">sax@megginson.com</a>
       * @version 2.0
       * @see org.xml.sax.Parser#setErrorHandler
       * @see org.xml.sax.SAXParseException
       */
      public interface ErrorHandler {


          /**
           * Receive notification of a warning.
           *
           * <p>SAX parsers will use this method to report conditions that
           * are not errors or fatal errors as defined by the XML 1.0
           * recommendation.  The default behaviour is to take no action.</p>
           *
           * <p>The SAX parser must continue to provide normal parsing events
           * after invoking this method: it should still be possible for the
           * application to process the document through to the end.</p>
           *
           * <p>Filters may use this method to report other, non-XML warnings
           * as well.</p>
           *
           * @param exception The warning information encapsulated in a
           *                  SAX parse exception.
           * @exception org.xml.sax.SAXException Any SAX exception, possibly
           *            wrapping another exception.
           * @see org.xml.sax.SAXParseException
           */
          public abstract void warning (SAXParseException exception)
            throws SAXException;


          /**
```

```
     * Receive notification of a recoverable error.
     *
     * <p>This corresponds to the definition of "error" in section 1.2
     * of the W3C XML 1.0 Recommendation.  For example, a validating
     * parser would use this callback to report the violation of a
     * validity constraint.  The default behaviour is to take no
     * action.</p>
     *
     * <p>The SAX parser must continue to provide normal parsing events
     * after invoking this method: it should still be possible for the
     * application to process the document through to the end.  If the
     * application cannot do so, then the parser should report a fatal
     * error even if the XML 1.0 recommendation does not require it to
     * do so.</p>
     *
     * <p>Filters may use this method to report other, non-XML errors
     * as well.</p>
     *
     * @param exception The error information encapsulated in a
     *                  SAX parse exception.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see org.xml.sax.SAXParseException
     */
    public abstract void error (SAXParseException exception)
      throws SAXException;


    /**
     * Receive notification of a non-recoverable error.
     *
     * <p>This corresponds to the definition of "fatal error" in
     * section 1.2 of the W3C XML 1.0 Recommendation.  For example, a
     * parser would use this callback to report the violation of a
     * well-formedness constraint.</p>
     *
     * <p>The application must assume that the document is unusable
     * after the parser has invoked this method, and should continue
     * (if at all) only for the sake of collecting addition error
     * messages: in fact, SAX parsers are free to stop reporting any
     * other events once this method has been invoked.</p>
     *
     * @param exception The error information encapsulated in a
     *                  SAX parse exception.
     * @exception org.xml.sax.SAXException Any SAX exception, possibly
     *            wrapping another exception.
     * @see org.xml.sax.SAXParseException
     */
    public abstract void fatalError (SAXParseException exception)
      throws SAXException;

}

// end of ErrorHandler.java
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. THE SAAM CONFIGURATION DTD FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Mohammad Ababneh (U.S. Naval Postgraduate School/Computer Science) -->
<!--DTD generated by XML Spy v3.0.7 NT (http://www.xmlspy.com)-->
<!-- The DTD file enforces the rules under which the XML file is composed. It defines the structure of the XML file.
 If any rule is broken by the XML file then that file will be invalid.
XMLSPY validates the file and decides if it conforms to the rules, but the user has the choice to proceed
without validity. In that case unexpectable rsults can happen. In SAAM configuration XML file a non-valid f
ile will cause a parsing error and the configuration information will not be extracted from the file.-->

<!-- The rules inforced by the DTD are :
1· Each node in SAAM requires some elements to be mandatory. NodeType, NodeName, IPv4 and TimeScale
   are mandatory  and there should be only one of these elements for servers and routers.
2· Some of them can be included in the Server node, but not the Router node. SC_GlobalWaitTime,
   SC_LocalWaitTime,    SC_CycleTime and SC_MetricType are exclusively for the servers which
   are either Primary or Backup. Only one value of    each element should be included for a Server node.
   These values are required to instantiate a server, but they are not   required by routers. Thus, a node can
   have zero or one of these elements.
3· All nodes are required to have at least one interface element or it cannot participate in any topology.
   Thus, for each SAAM   node, there should be one or more Interface elements.
4· Each Interface element should have one Address element and one MaskBits element.
5· Each node can have zero or more Agent elements.

These rules are rpresented by the following DTD syntax

Operator      Description
No Operator       Must appear exactly one time
?             Must appear once or not at all
+             Must appear at least once ( 1 ... N times)
*             May appear any number of times, or not at all ( 0 ... N times)
-->


 <!-- PCDATA stands for Parced Character Data and it means that the element contains text data " -->
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Agent (#PCDATA)>
<!ELEMENT IPv4 (#PCDATA)>
<!ELEMENT Interface (Address, MaskBits)>
<!ELEMENT InterfaceAddressType (#PCDATA)>
<!ELEMENT MaskBits (#PCDATA)>
<!ELEMENT Node (NodeType, NodeName, IPv4, ServerFlowID?, TimeScale, SC_MetricType?, SC_CycleTime?, SC_GlobalWaitTime?, SC_LocalWaitTime, InterfaceAddressType, Interface+, Agent*)>
<!ELEMENT NodeName (#PCDATA)>
<!ELEMENT NodeType (#PCDATA)>
<!ELEMENT SAAM_Net (Node+)>
<!ELEMENT SC_CycleTime (#PCDATA)>
<!ELEMENT SC_GlobalWaitTime (#PCDATA)>
<!ELEMENT SC_LocalWaitTime (#PCDATA)>
<!ELEMENT SC_MetricType (#PCDATA)>
<!ELEMENT ServerFlowID (#PCDATA)>
<!ELEMENT TimeScale (#PCDATA)>
```

APPENDIX F. THE XML CONFIGURATION FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Geoffrey Xie (U.S. Naval Postgraduate School/Computer Science) -->
<!DOCTYPE SAAM_Net SYSTEM "file:///C:/XML/SAAM_1.dtd">

<!-- The XML file contains the configuration information of the desired network topology. It makes sure that the structure is correct by
vakidating these contents with the DTD file, which is defined above

The SAAM configuration consists from elements which can contain other elements.The  main element is the node which represents
the servers and routers. each node has a set of elements and values.-->
<SAAM_Net>
  <Node>
    <NodeType>PrimaryServer</NodeType>
    <NodeName>Server A</NodeName>
    <IPv4>131.120.9.46</IPv4>
    <ServerFlowID>1</ServerFlowID>
    <TimeScale>350</TimeScale>
    <!--SC = Self Configuration-->
    <SC_MetricType>0</SC_MetricType>
    <SC_CycleTime>200</SC_CycleTime>
    <SC_GlobalWaitTime>250</SC_GlobalWaitTime>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
      <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1</Address>
      <MaskBits>40</MaskBits>
    </Interface>
  </Node>
  <Node>
    <NodeType>Router</NodeType>
    <NodeName>Router A</NodeName>
    <IPv4>131.120.8.155</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
      <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2</Address>
      <MaskBits>40</MaskBits>
    </Interface>
    <Interface>
      <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.3</Address>
      <MaskBits>40</MaskBits>
    </Interface>
    <Agent>PreviousNodeProbe</Agent>
  </Node>
  <Node>
    <NodeType>Router</NodeType>
    <NodeName>Router B</NodeName>
    <IPv4>131.120.9.49</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
      <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.4</Address>
      <MaskBits>40</MaskBits>
    </Interface>
    <Agent>PreviousNodeProbe</Agent>
    <Agent>NextNodeProbe</Agent>
  </Node>
</SAAM_Net>
```

APPENDIX G. A USER GUIDE TO USE XMLSPY IN SAAM CONFIGURATION

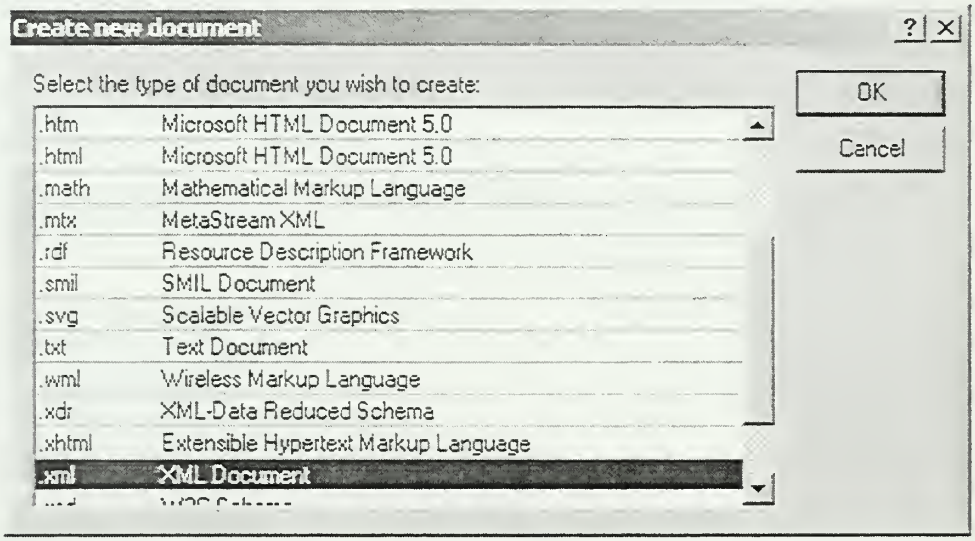
A QUICK USER GUIDE FOR USING XMLSPY 3.0 IN SAAM CONFIGURATION

In this short tutorial we will learn how to configure the SAAM topology using the newly developed XML approach. XMLSPY 3.0 is a very powerful tool that makes the creation and modification of any topology very easy and user friendly.

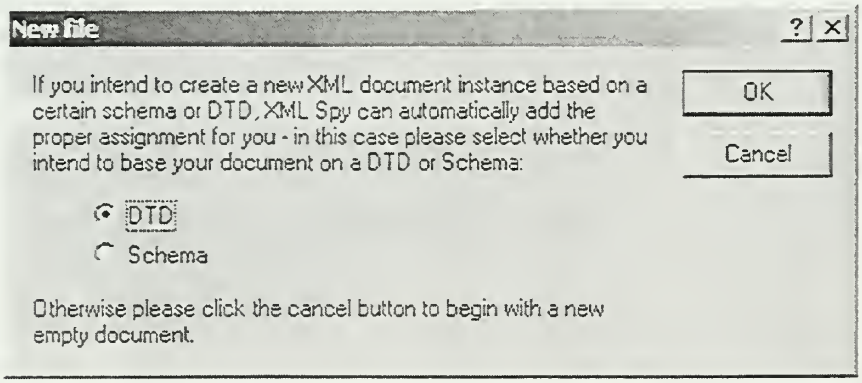
A. XML and DTD operation.

1. Creating a new XML file

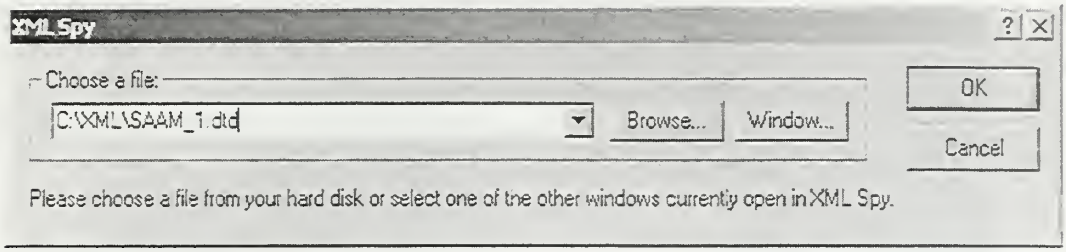
To create a new XML file for the purpose of SAAM configuration, go to XMLSPY "File – New", which causes this dialogue box to appear.



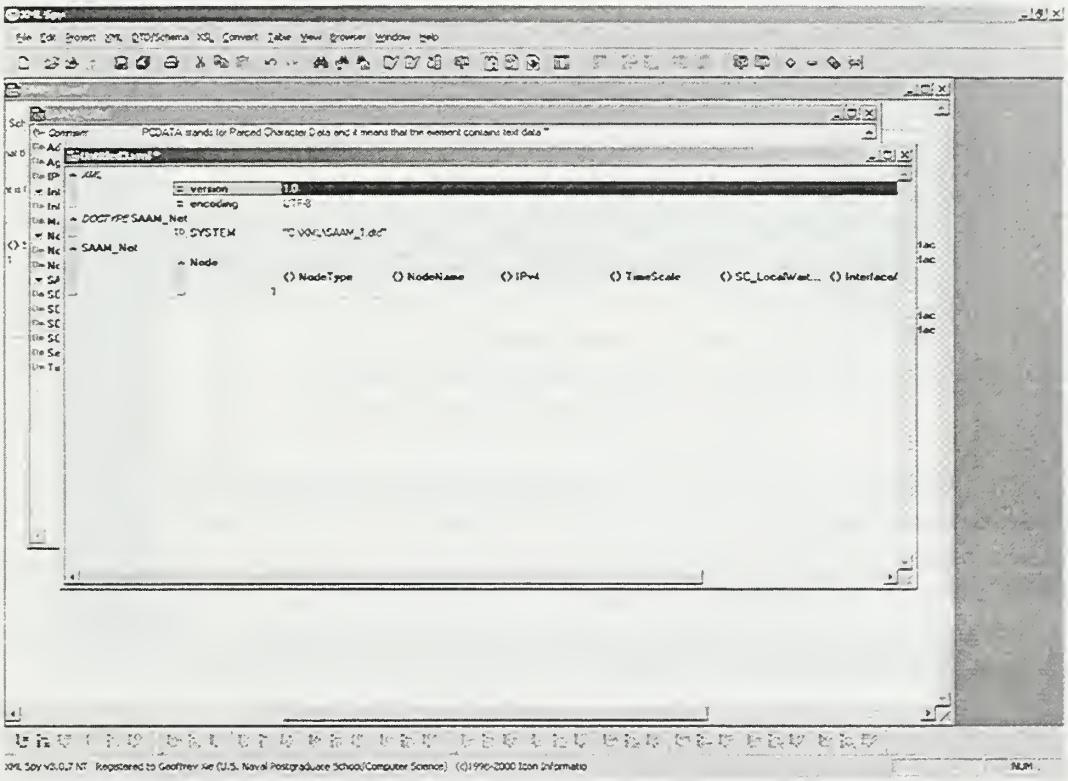
Select the XML Document and click "OK". The IDE will display the following dialogue box, which asks weather are we going to use a DTD file or a schema.


New file
If you intend to create a new XML document instance based on a certain schema or DTD, XML Spy can automatically add the proper assignment for you - in this case please select whether you intend to base your document on a DTD or Schema:
OK
Cancel
DTD
Schema
Otherwise please click the cancel button to begin with a new empty document.

We will select DTD and the next dialogue will appear to insert the path of your DTD file.



Now click Ok and the following window will open with all the elements predefined in the DTD file. Start filling the values and insert the number of rows that represents your nodes. One row represents one node.



2. **Updating an XML file**
   a. **Add a node**

      To add a node to an XML file, just click "Table – Insert Row" or "Table – Append Row".

   b. **Delete a node**

      To delete a node, just select the row that represents the node and click "Delete" on the keyboard.

   c. **Update a node**

      To update the values of the node elements, take the cursor on the cell that represents that element's value and inert in your new value.

3. **Modifying the structure of an XML file**

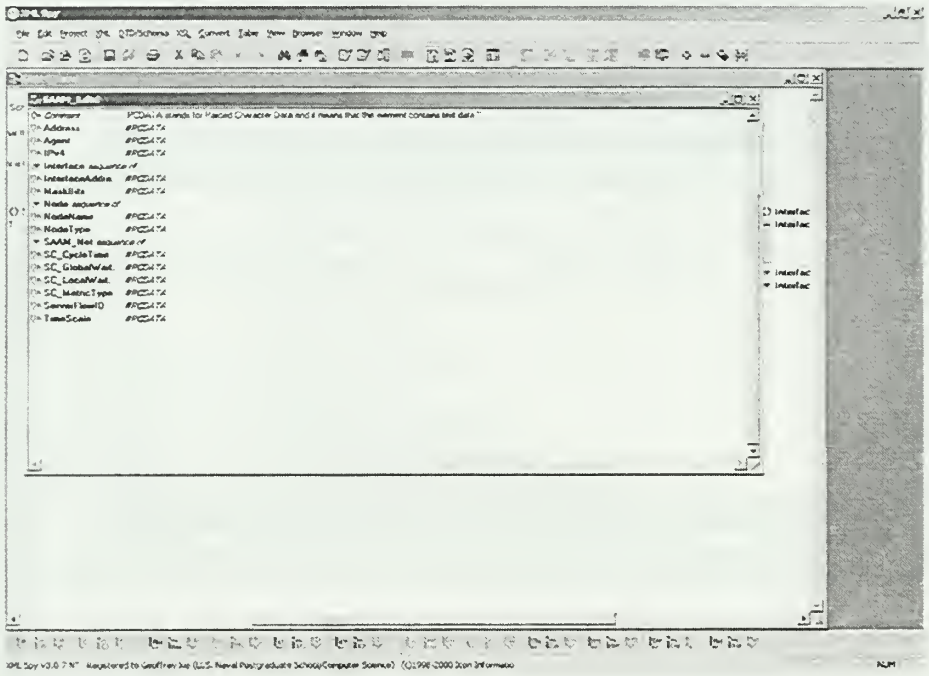
Here we mean that you can add a new element with its value to the XML file. This might be needed in order to define new service metrics or configuration elements or to delete some existing elements.

In order to make this modification, we should start with modifying the DTD file. Otherwise, we will get an invalid XML file.

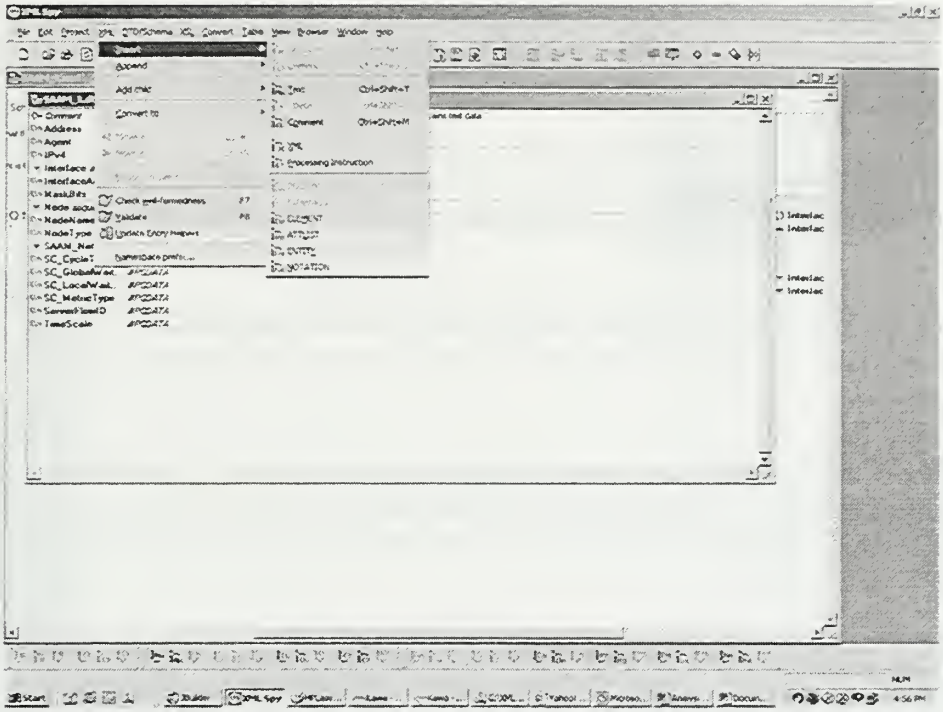
So, the steps to modify the structure of an XML file are:

1. **Step One:**

Open the DTD file and switch to the "Enhanced Grid View". While you are in the tabular view, select the row that you would like to make modifications on.



2. **Step Two:**
   a. To add a new element click "XML – Insert – ELEMENT". Fill in the element name and its data type.



b. To delete an element select the line and click delete.
c. To change the element name or the data type, just click on the name or type and insert the new value

B. Validation and well-formedness

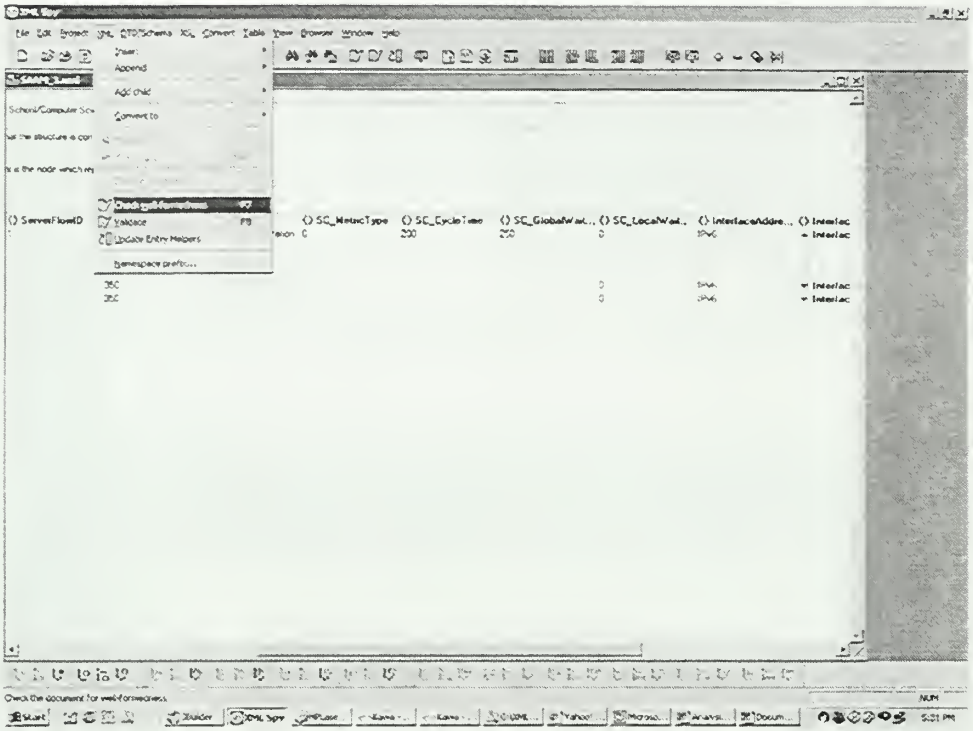
In order to be able to use and XML with the Java parser program, that file should be valid and well-formed. Well-formedness means that the XML is not violating any XML syntax rules. On the other hand, the purpose of validation is to make sure that the structure of the XML document conforms to the rules enforced by the DTD file.

It is necessary to check well-formedness and validity each time a single change is made to the XML or the DTD file.

XMLSPY 3.0 supports a powerful check for these two properties of XML:

1. Well-formedness

To check weather a file is well-formed, click go to the XML menu and select "Check well-formedness". If anything is wrong a warning is displayed.



2. Validity

To check weather a file is valid, click go to the XML menu and select "Validate". This validiation process requires the user to specify the DTD file previously when he creates the file. If anything is wrong a warning is displayed.

LIST OF REFERENCES

1. Quek, Henry, "QoS Management With Adaptive Routing For Next Generation Internet", Thesis March 2000, Computer Science Department Naval Postgraduate School.

2. Vrable, Dean, Yarger, John, "The SAAM Architecture: Enabling Integrated Services", Thesis September 1999, Computer Science Department Naval Postgraduate School.

3. Xie, Geoffrey G., Hensgen, Debra, Kidd, Taylor, and Yarger, John, "SAAM: An Integrated Network Architecture for Integrated Services," paper presented at the 6th IEEE/IFIP International Workshop on Quality of Service, Napa, CA, May 1998.

4. Ababneh, Mohammad, Brunstad, Dag-Anders, Gaines Len, " An ASCII Approach for Configuring SAAM Networks", CS4552 Network programming class Project June 2000, Computer Science Department Naval Postgraduate School.

5. Marchal, Benoit, *XML by Example*, 1st edition, Que Education and Training, December 1999..

6. The World Wide Web Consortium web site, [http://www.w3.org/TR/REC-xml], September 2000.

7. McLaughlin, Bert, *Java and XML,* O'reilly, June 2000.

8. Akkoc, Hasan, "A Pro-Active Routing Protocol for Configuration Signaling Channels in SAAM", Thesis June 2000, Computer Science Department Naval Postgraduate School.

9. The Apache web site, [http://www.apache.org], September 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Information Systems Academic Group 1
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Prof. Geoffrey Xie, Code CS/Xg 1
   Naval Postgraduate School
   Monterey, CA 93943-5100

5. Prof. Daniel Dolk, Code IS/DK 1
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. Mr. Cary Colwell 1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93943-5100

7. Mohammad Ababneh 5
   Royal Jordanian Air Force
   Information and Computer Department
   Amman - Jordan

72 290NPG TH 3180
6/02 22527-200 NLE

DUDLEY KNOX LIBRARY
3 2768 00403020 5